KIG4068 : Machine Learning

Week 2: End-to-End Machine Learning Project

Semester 2, Session 2023/2024

This Week's Lesson Overview

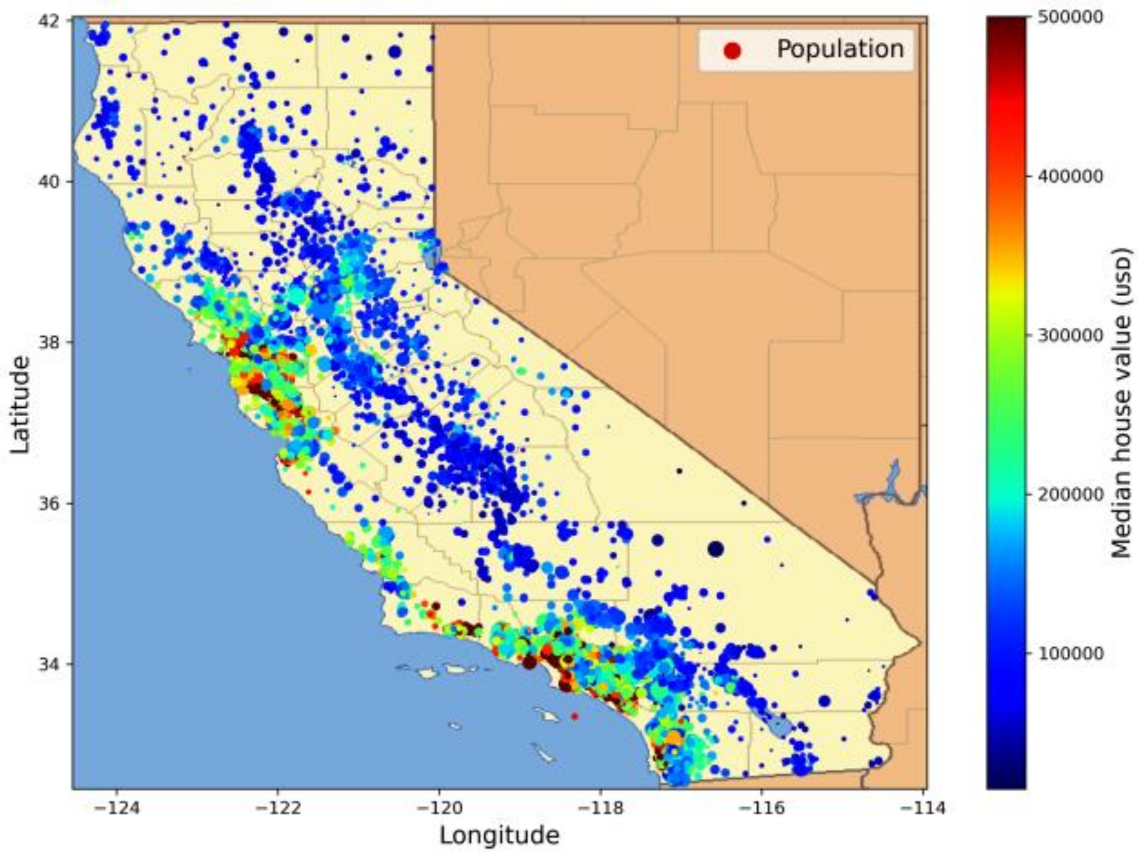
Main steps that we will walk through, to tackle a machine learning project:

1) Look at the big picture.
2) Get the data.
3) Explore 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 the model.
7) Present the solution.
8) Launch, monitor and maintain the system.

Look at the Big Picture

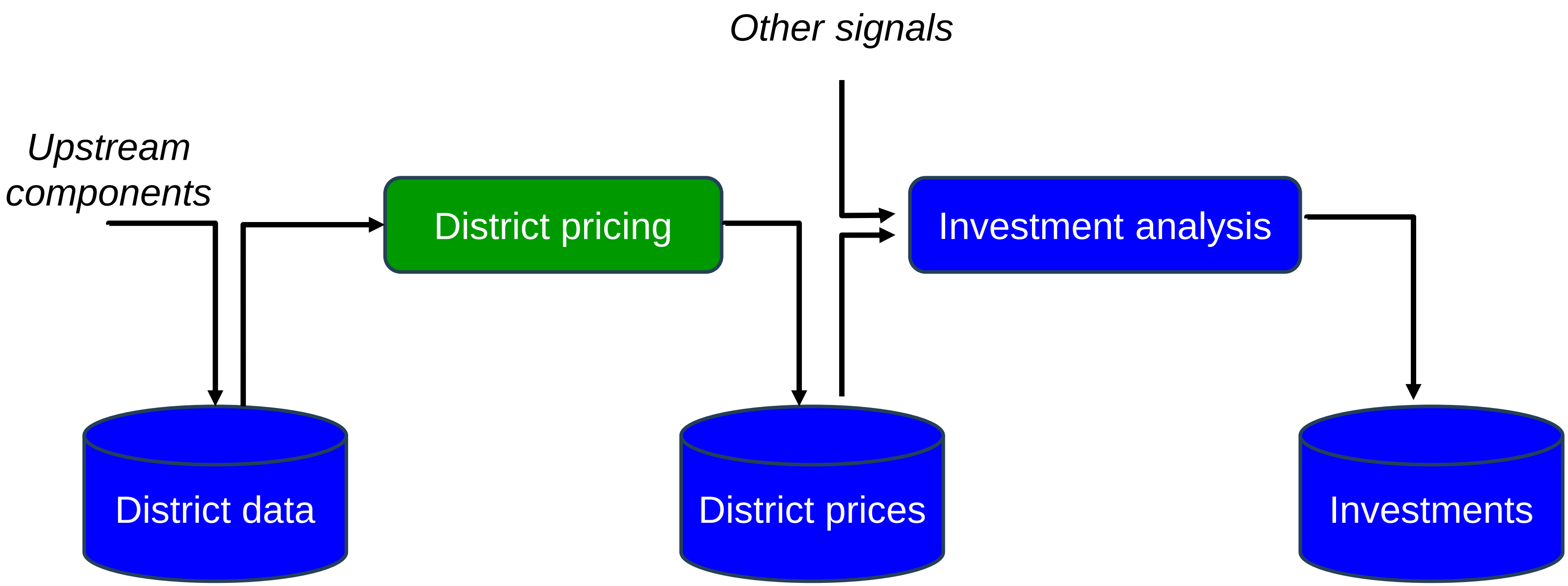
Task: To build a model to predict the median housing price in any district.

Data parameters:
- Population
- Median income
- Median housing price
- Districts
- Median age
- Total rooms
- Total bedrooms



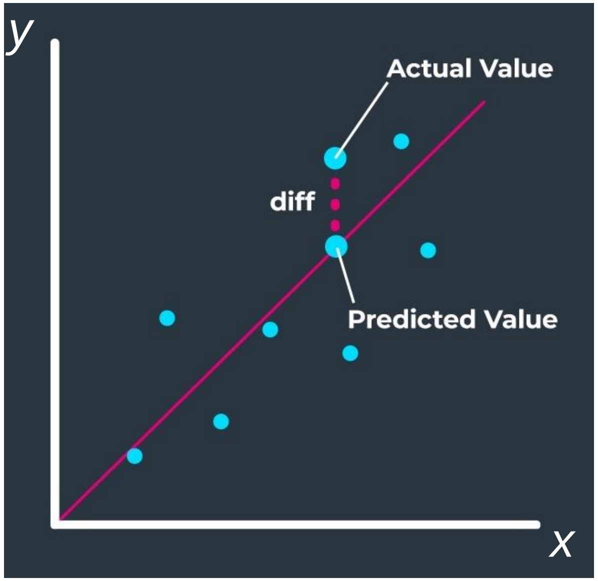
California housing prices

Frame the Problem



Questions to ask	Modelling approach
What is the model objectives & expectations?	What kind of training supervision the model will need?
What the current solution looks like?	Is it a classification task, regression task?
	Should it be a batch or online learning?

Select a Performance Measure



Mean Absolute Error (MAE)

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i|$$

OR

Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

Where: n is the number of observations
y_i is the actual value
$\hat{y}_i$ is the predicted value

Load the Data

```
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://yourURLhere.com/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
        with tarfile.open(tarball_path) as housing_tarball:
            housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

Download data from a URL, and load the data into Pandas DataFrame

Load the data into Pandas DataFrame from a folder on your computer

```
from pathlib import Path
import pandas as pd
import numpy as np

def load_housing_data():
    filepath = "C:/Users/chest/OneDrive - Universiti Malaya/UM/Teaching/KIG4068 Machine Learning/Data/"
    return pd.read_csv(Path(filepath+"datasets/housing/housing.csv"))

housing = load_housing_data()
```

Data Structure

df.head()/ df.tail() –Displays the first/ last few rows of DataFrame

df.info() – Provides a concise summary of the DataFrame (entries, column names etc.)

df.describe() – Generates descriptive statistics of the numerical column (count, mean, std etc.)

df.hist() – Creates histograms for numerical columns in a DataFrame

Example:

housing.head()

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

Data Structure (Cont'd)

housing.info()

```
<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: 1.6+ MB
```

Total number of instances/ rows

Some rows in this column contain null values

Data type

housing["ocean_proximity"].value_counts()

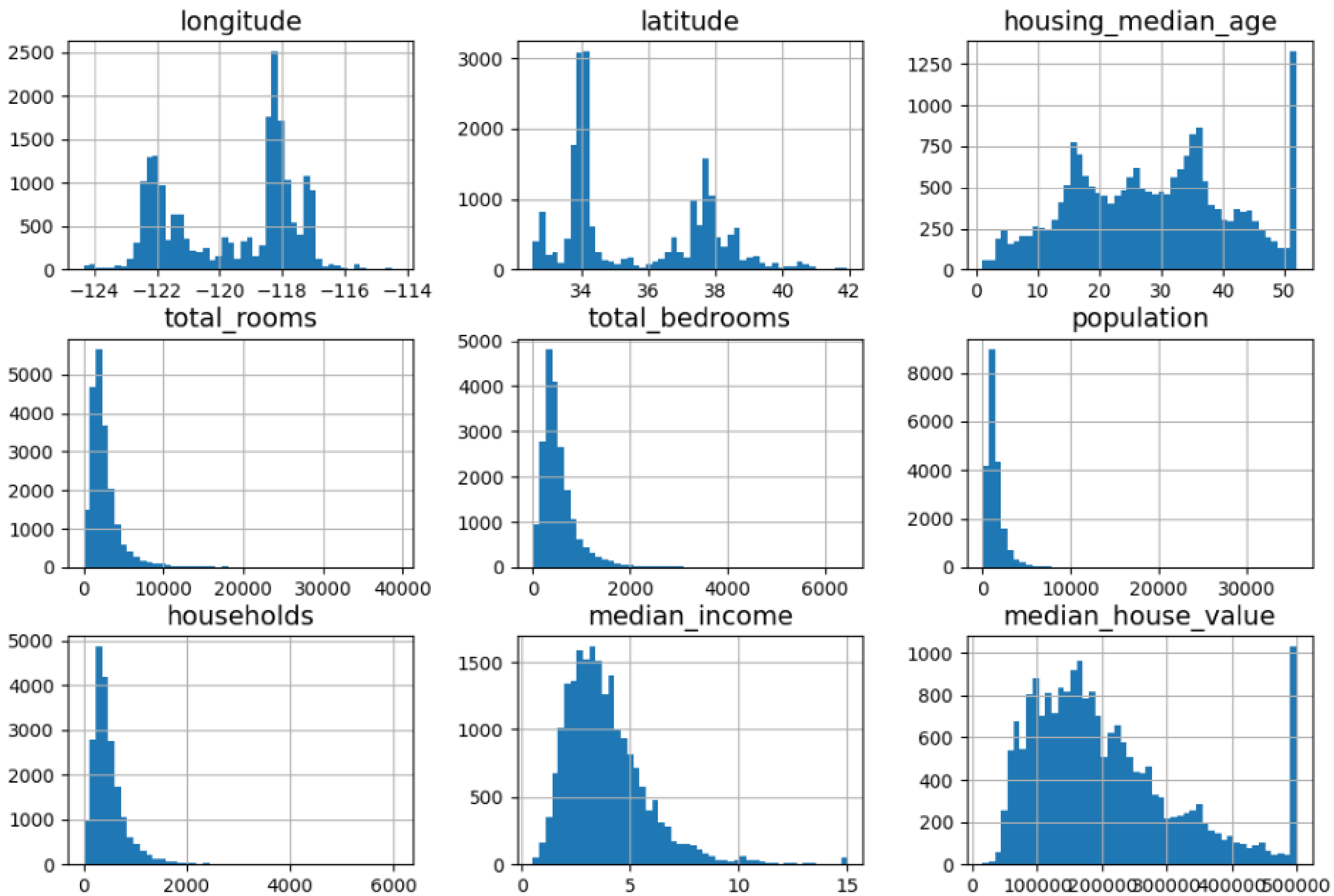
```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

housing.describe()

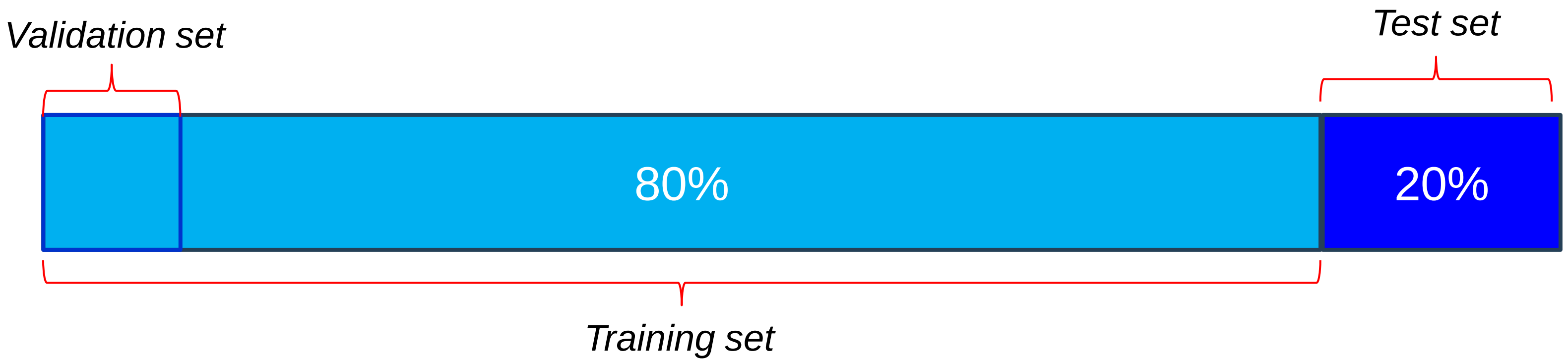
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Data Structure (Cont'd)

housing.hist(bins, figsize, etc)



Creating a Test Set



Training set

- Portion of the dataset used to train ML model.
- Consists of input data points (features) with their corresponding target labels/ outcomes.
- The model learns from the patterns and relationship present in the training data to make predictions.

Validation set

- Used to tune hyperparameters and evaluate performance during training.
- Helps in selecting best model architecture and parameter settings.
- Provides unbiased estimate because it is not used during training.

Test set

- Independent dataset that is not used during training or tuning.
- Used to assess the performance of the trained model on unseen data.
- Helps to estimate how well the model will perform in real-world scenarios.

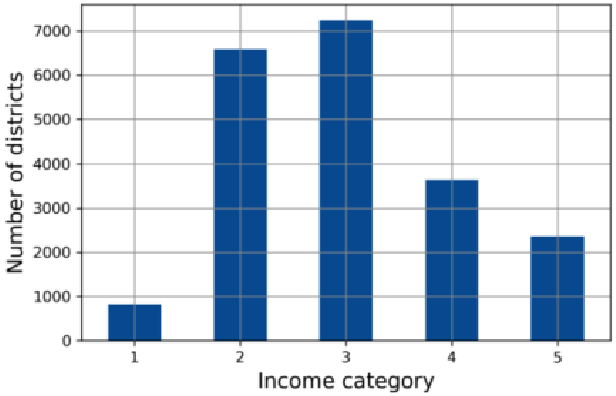
Creating a Test Set (Cont'd)

Option 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)
```

Option 2 (stratified sampling)

```
housing["income_cat"] = pd.cut(housing["median_income"],
                               bins=[0., 1.5, 3.0, 4.5, 6., np.inf],
                               labels=[1, 2, 3, 4, 5])
```

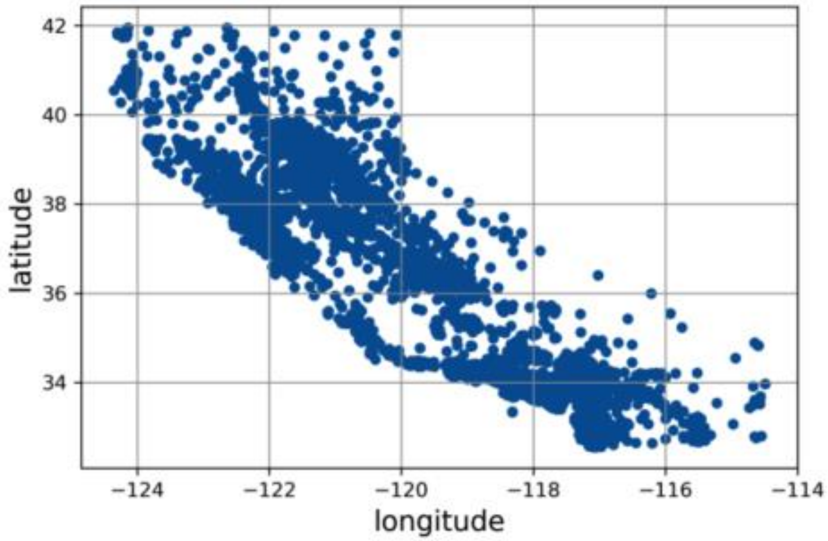



```
housing["income_cat"].value_counts().sort_index().plot.bar(rot=0, grid=True)
plt.xlabel("Income category")
plt.ylabel("Number of districts")
plt.show()
```

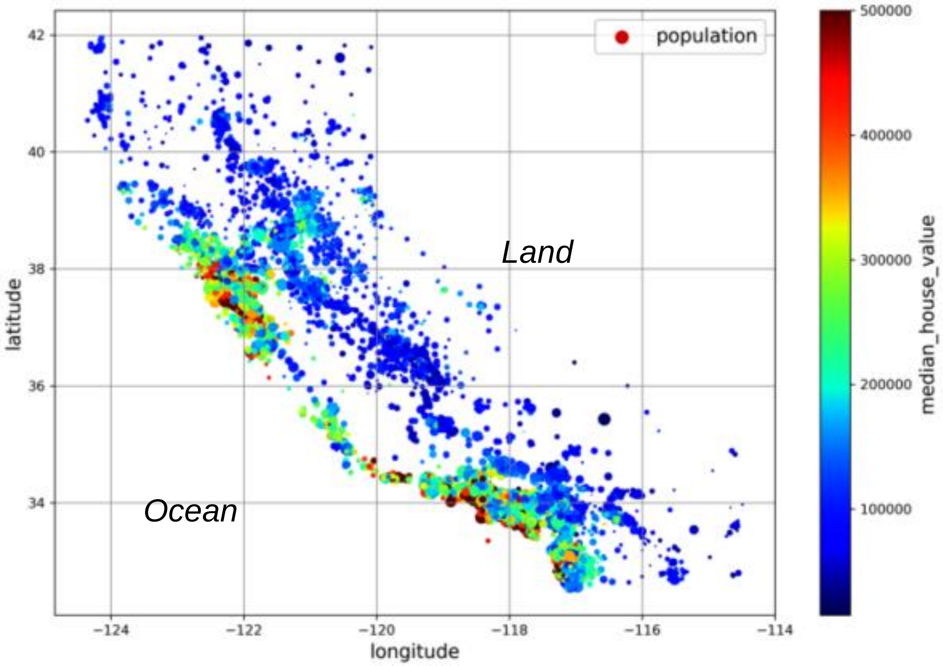
```
strat_train_set, strat_test_set = train_test_split(
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

Data Visualization & Exploration

```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
plt.show()
```




```
housing.plot(kind="scatter", x="longitude", y="latitude", grid=True,
             s=housing["population"] / 100, label="population",
             c="median_house_value", cmap="jet", colorbar=True,
             legend=True, sharex=False, figsize=(10, 7))
plt.show()
```



Data Visualization & Exploration (Cont'd)

Standard correlation coefficient (Pearson's r)

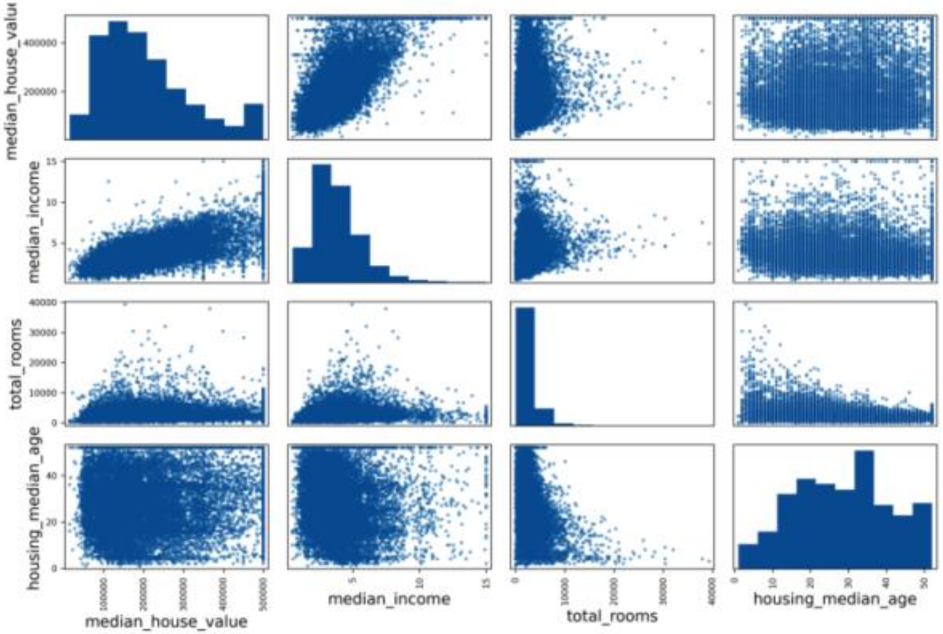
```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix["median_house_value"].sort_values(ascending=False)
```

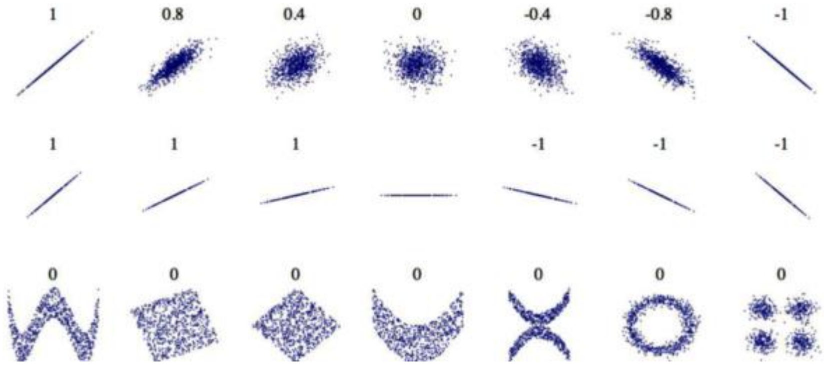
```
median_house_value    1.000000
median_income         0.688380
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
longitude            -0.050859
latitude             -0.139584
Name: median_house_value, dtype: float64
```

Data Visualization & Exploration (Cont'd)

Plotting correlation between attributes using Pandas

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms",
              "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```





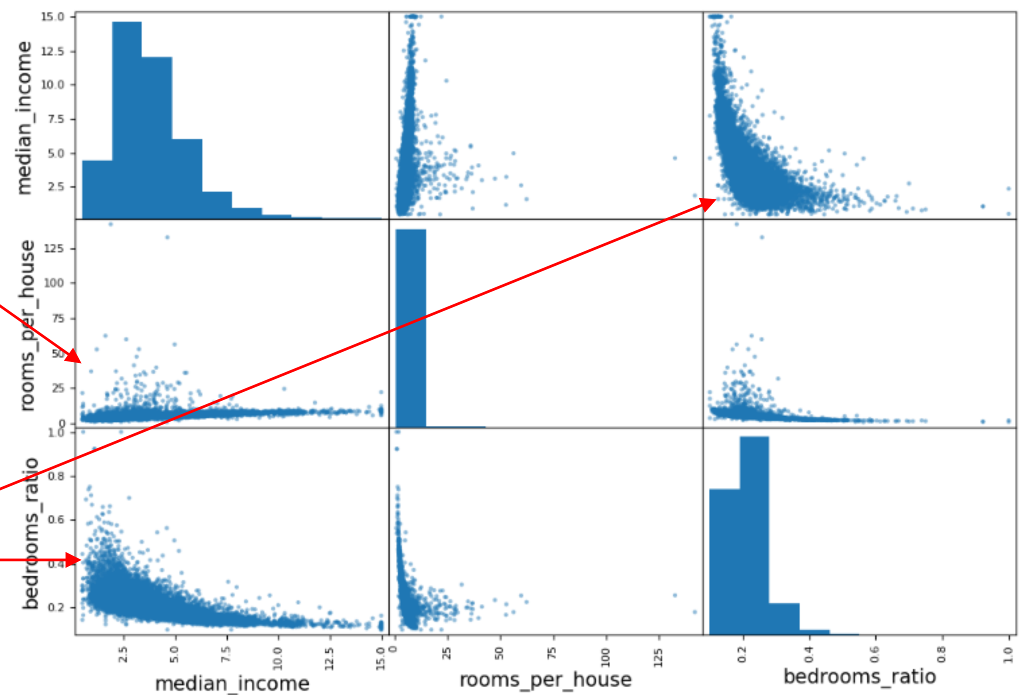
Standard correlation coefficient of various datasets

Data Manipulation

Experiment with attribute combinations

```
housing["rooms_per_house"] = housing["total_rooms"] / housing["households"]
housing["bedrooms_ratio"] = housing["total_bedrooms"] / housing["total_rooms"]
housing["people_per_house"] = housing["population"] / housing["households"]
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

```
median_house_value    1.000000
median_income         0.688380
rooms_per_house       0.143663
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population           -0.020153
people_per_house     -0.038224
longitude            -0.050859
latitude             -0.139584
bedrooms_ratio       -0.256397
Name: median_house_value, dtype: float64
```



Data Cleaning

Options to work with missing values

Option 1 (Remove the rows of missing values)

```
housing.dropna(subset=["total_bedrooms"], inplace=True)  # option 1
```

Option 2 (Remove the whole attribute or column)

```
housing.drop("total_bedrooms", axis=1)  # option 2
```

Option 3 (Imputation – set missing values to some values, e.g. zero, mean, median etc)

```
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

OR

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy="median")

housing_num = housing.select_dtypes(include=[np.number])

imputer.fit(housing_num)

X = imputer.transform(housing_num)

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

Use scikit-learn imputation function

Data Cleaning (Cont'd)

Convert text to numbers

```
housing_cat = housing[["ocean_proximity"]]
housing_cat.head(8)
```

	ocean_proximity
13096	NEAR BAY
14973	<1H OCEAN
3785	INLAND
14689	INLAND
20507	NEAR OCEAN
1286	INLAND
18078	<1H OCEAN
4396	NEAR BAY

Option 1 (Ordinal Encoder)

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)

housing_cat_encoded[:8]

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

```
ordinal_encoder.categories_

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

Option 2 (One-hot Encoder)

```
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)

housing_cat_1hot.toarray()

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

cat_encoder.categories_

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

Feature Scaling & Transformation

Options to work with different attributes scales

Option 1 (Normalization)

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)
```

$$X_{norm} = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

Option 2 (standardization)

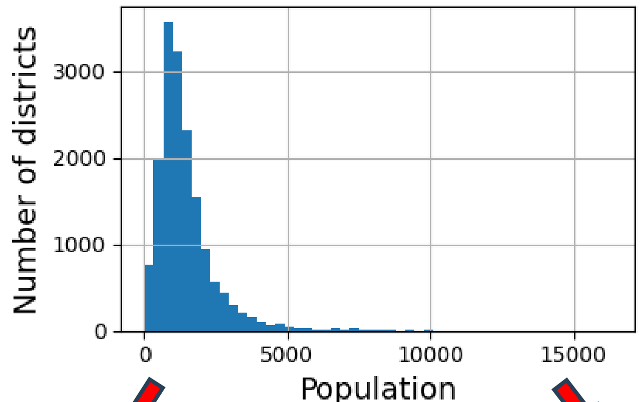
```
from sklearn.preprocessing import StandardScaler

std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```

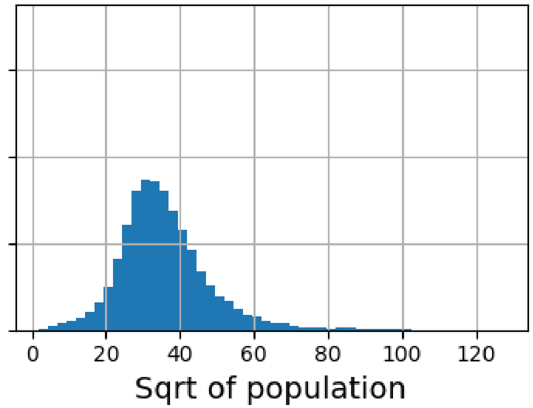
$$X_{std} = \frac{(X - mean)}{std}$$

Normalization	Standardization
Rescales values to (e.g.,-1, 1 or 0,1)	Not bounded to a certain range
Useful when data distribution is unknown or not Gaussian	Useful when data distribution is Gaussian or unknown
Sensitive to outliers	Less sensitive by outliers
Retains the shape of the original distribution	Changes the shape of the original distribution
May not preserve the relationship between data points	Preserves the relationships between data points

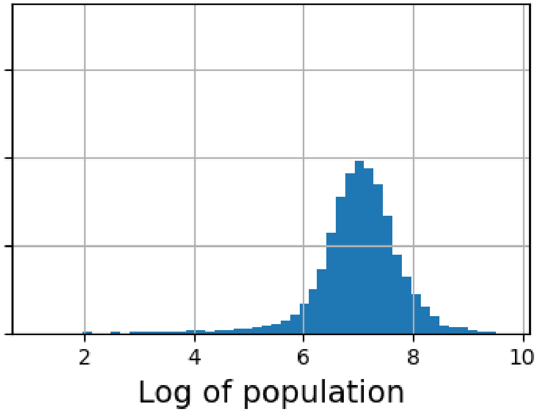
Feature Scaling & Transformation (Cont'd)

Pre-feature scaling transformations





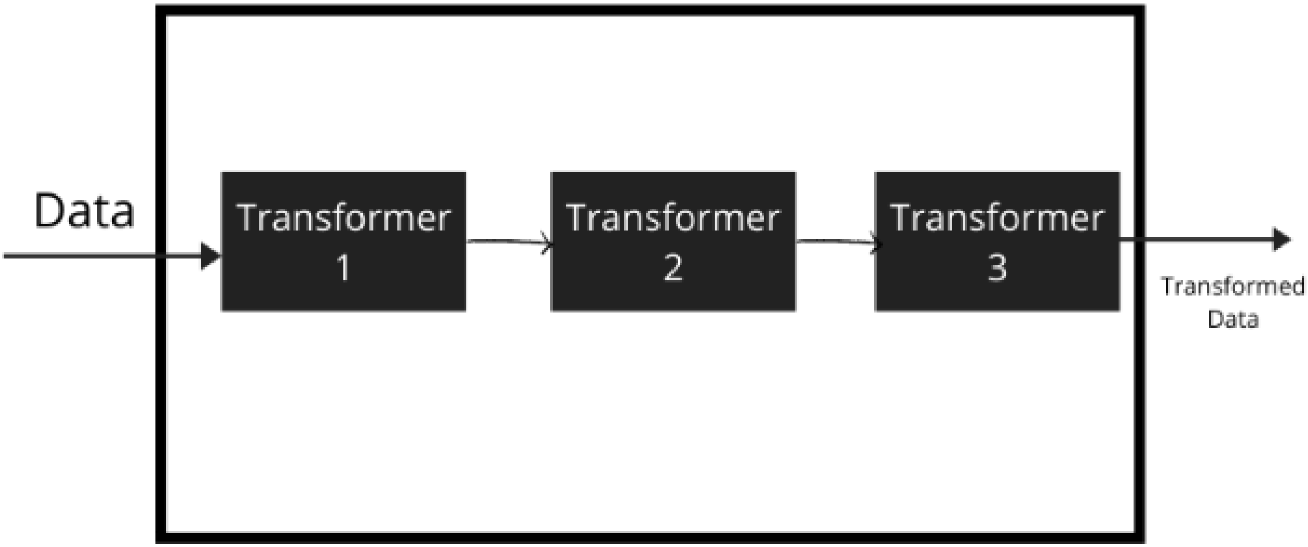
Option 1 (replace the feature with its square root)



Option 1 (replace the feature with its logarithm)

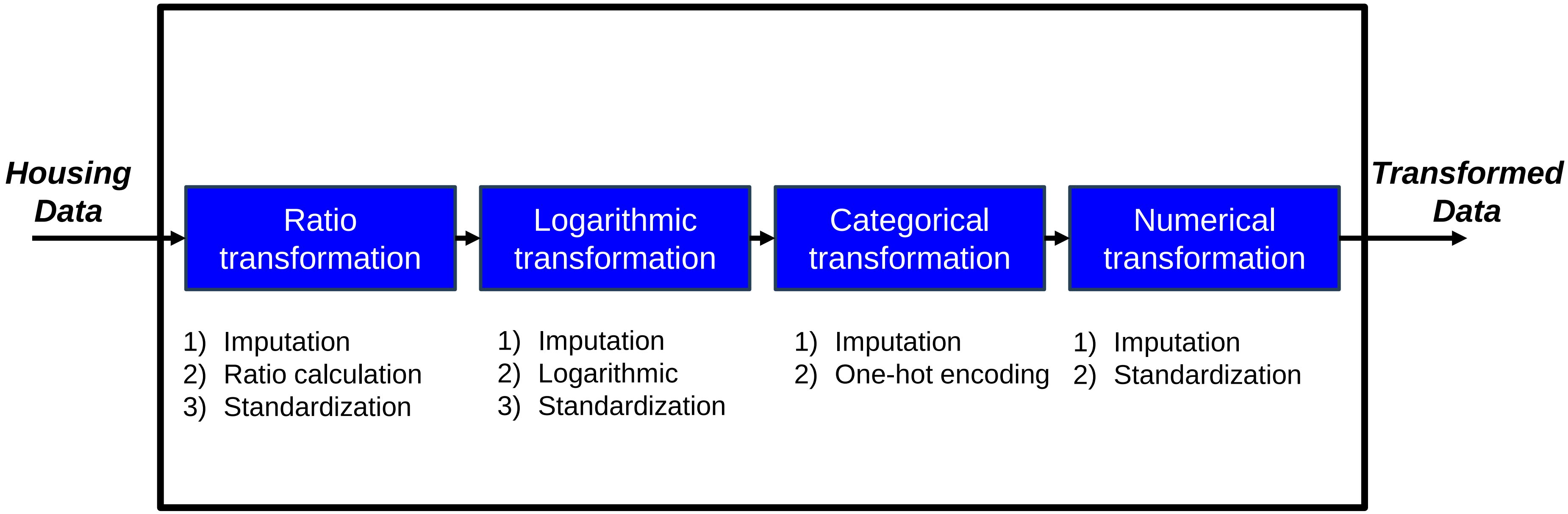
Transformation Pipeline

- Machine learning pipelines are mechanism that chains multiple steps together, ensuring that the output of each step is used as input to the next step.
- In other words, a machine learning pipeline performs a sequence of steps, where the output of the first transformer becomes the input for the next transformer.



Transformation Pipeline (Cont'd)

<u>Example: Housing data</u>



Transformation Pipeline (Cont'd)

```python
from sklearn.pipeline import make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import FunctionTransformer
from sklearn.compose import make_column_selector

def ratio_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(column_ratio, feature_names_out=ratio_name),
        StandardScaler())

def column_ratio(X):
    return X[:, [0]] / X[:, [1]]

def ratio_name(function_transformer, feature_names_in):
    return ["ratio"] # feature names out

def log_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        FunctionTransformer(np.log, feature_names_out="one-to-one"),
        StandardScaler())

def cat_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="most_frequent"),
        OneHotEncoder(handle_unknown="ignore"))

def default_num_pipeline():
    return make_pipeline(
        SimpleImputer(strategy="median"),
        StandardScaler())
```

```python
preprocessing = ColumnTransformer([
("bedrooms", ratio_pipeline(),
 ["total_bedrooms", "total_rooms"]),
("rooms_per_house", ratio_pipeline(),
 ["total_rooms", "households"]),
("people_per_house", ratio_pipeline(),
 ["population", "households"]),
("log", log_pipeline(),
 ["total_bedrooms", "total_rooms", "population",
"households", "median_income"]),
("cat", cat_pipeline(),
 make_column_selector(dtype_include=object)),
],remainder=default_num_pipeline())
# remaining col: housing_median_age

housing_prepared = preprocessing.fit_transform(housing)
housing_prepared.shape
```

- **ColumnTransformer**: This allow us to apply different transformation to different columns
- **FunctionTransformer**: Creates a transformer from an arbitrary function (custom function)
- **make_pipeline**:Creates a pipeline by concentrating multiple transformers
- **make_column_selector**: creates a colun selector on specified criteria

Select & Train the Model

Example 1 – Linear Regression Model

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import root_mean_squared_error
lin_reg = make_pipeline(preprocessing, LinearRegression())
lin_reg.fit(housing, housing_labels)
housing_predictions = lin_reg.predict(housing)
lin_rmse = mean_squared_error(housing_labels, housing_predictions,squared=False)
```

RMSE = 70632

Example 2 – Decision Tree Model

```
from sklearn.tree import DecisionTreeRegressor
tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))
tree_reg.fit(housing, housing_labels)
housing_predictions = tree_reg.predict(housing)
tree_rmse = mean_squared_error(housing_labels, housing_predictions,squared=False)
```

???

RMSE = 0

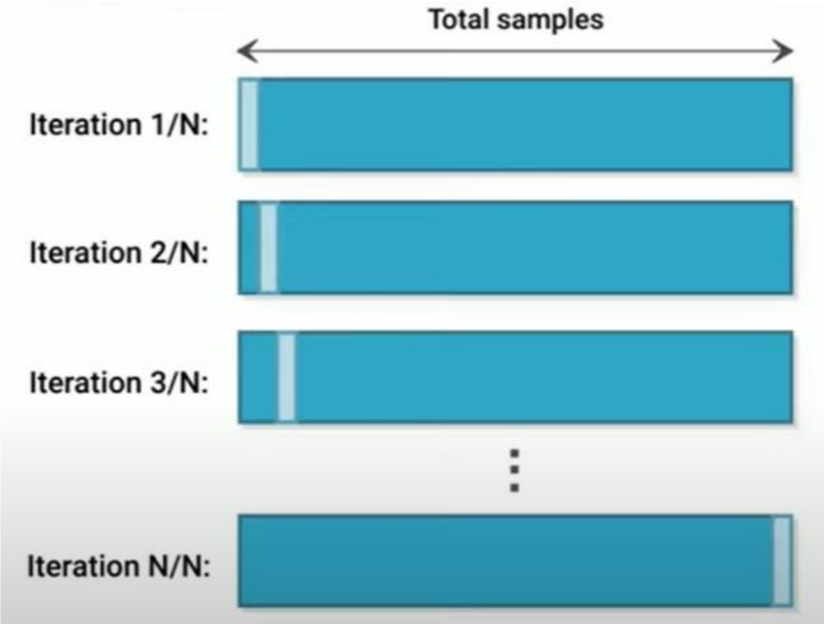
Example 3 – Random Forest Regressor Model

```
from sklearn.ensemble import RandomForestRegressor
forest_reg = make_pipeline(preprocessing,
                           RandomForestRegressor(random_state=42))
forest_reg.fit(housing, housing_labels)
housing_predictions = forest_reg.predict(housing)
forest_rmse = mean_squared_error(housing_labels, housing_predictions,squared=False)
```

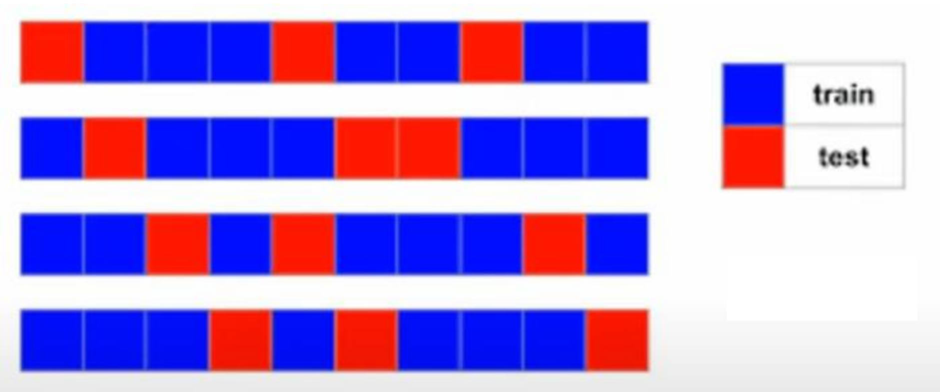
RMSE = 18469

Evaluation using Cross-Validation

- A family of techniques used to measure the effectiveness of predictions, generated from machine learning models.
- Some cross validation techniques:
    1. Leave-P-Out
    2. Leave-One-Out
    3. K-Fold
    4. Stratified K-Fold

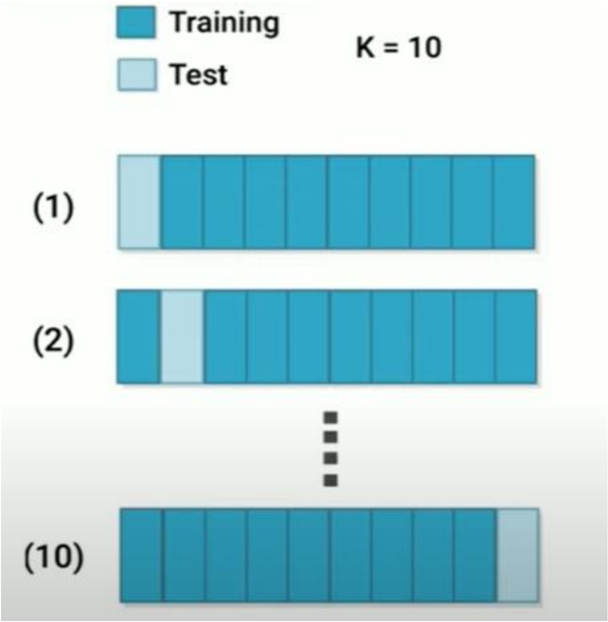


Leave-one-out cross validation

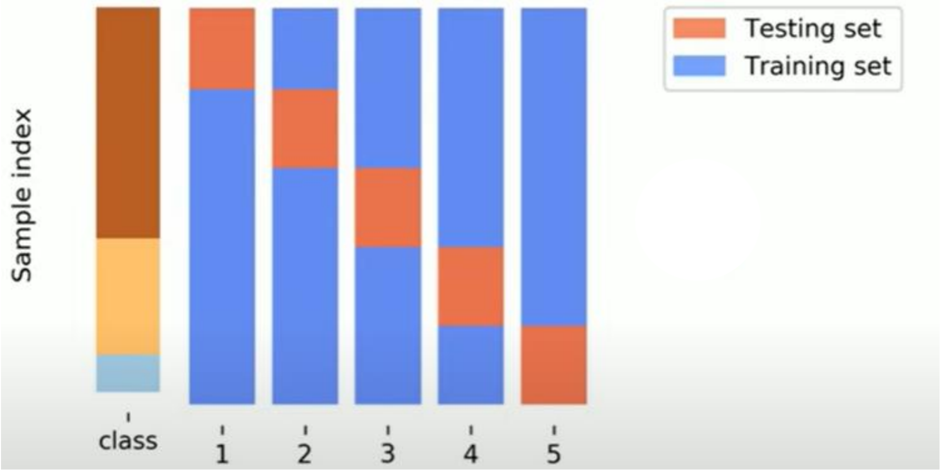


Leave-p-out cross validation

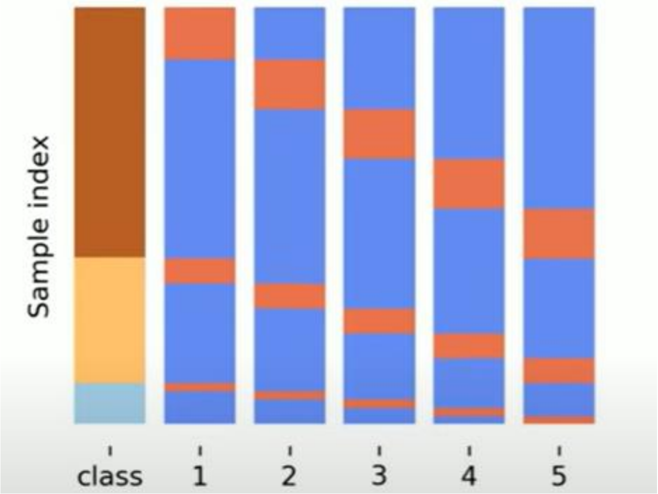
Evaluation using Cross-Validation



K-fold cross validation



K-fold cross validation



Stratified K-fold cross validation

Evaluation using Cross-Validation

Cross validation examples

K-fold cross validation (Example 2 – Decision Tree Model)

```
from sklearn.model_selection import cross_val_score
tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
scoring="neg_root_mean_squared_error", cv=10)
print("Cross-validation RMSEs (Kfold):", tree_rmses.mean())
```

```
Cross-validation RMSEs (Kfold): 70042.97685272685
```

Stratified K-fold cross validation (Example 2 – Decision Tree Model)

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
tree_rmses = -cross_val_score(tree_reg, housing, housing_labels,
                              scoring="neg_root_mean_squared_error", cv=skf)
print("Cross-validation RMSEs (Stratified Kfold):", tree_rmses.mean())
```

```
Cross-validation RMSEs (Stratified Kfold): 69535.43080443132
```

Fine Tuning the Model

- Hyperparameter tuning is a process of optimizing the hyperparameters of a machine learning model to improve its performance
- Common techniques for hyperparameter tuning include:
  1. **Grid Search**
     - Method: Exhaustively explores a predefined set of hyperparameter combinations.
     - Process: Evaluates each combination using cross – validation.
     - Outcome: Selects the combination with best performance.
  2. **Random Search**
     - Method: Randomly samples hyperparameter combinations from a specified distribution.
     - Advantage: Efficient for large search spaces.
     - Outcome: May not guarantee optimal combination but offers practical efficiency.
  3. **Bayesian Optimization**
     - Method: Utilizes probabilistic models to predict performance of hyperparameter configurations.
     - Approach: Iteratively selects new settings based on previous performance.
     - Goal: Aims for optimal configuration with fewer iterations.

Fine Tuning the Model

Hyperparameters tuning examples

```
from sklearn.pipeline import Pipeline
full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
```

Some other hyperparameters:
1) n_estimators
2) max_depth
3) min_samples_split
4) min_samples_leaf
5) bootstrap
6) max_samples

Option 1 – Grid Search

```
from sklearn.model_selection import GridSearchCV

param_grid = [
{'random_forest__max_features': [2, 4, 6, 8, 10, 12, 14, 16]}
]

grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
scoring='neg_root_mean_squared_error')

grid_search.fit(housing, housing_labels)
final_model = grid_search.best_estimator_
```

Option 2 – Random Search

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

param_distribs = {'random_forest__max_features': randint(low=2, high=20)}

rnd_search = RandomizedSearchCV(
full_pipeline, param_distributions=param_distribs, n_iter=10, cv=3,
scoring='neg_root_mean_squared_error', random_state=42)

rnd_search.fit(housing, housing_labels)
final_model2 = rnd_search.best_estimator_
```

Evaluate on the Test Set

Example 1 – Grid Search Model Evaluation

```
print("Random Forest Grid")
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
final_predictions = final_model.predict(X_test)
final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
print(final_rmse)
```

```
Random Forest Grid
48990.45202127383
```

Example 2 – Randomized Search Model Evaluation

```
print("Random Forest Randomized")
X_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()
final_predictions = final_model2.predict(X_test)
final_rmse = mean_squared_error(y_test, final_predictions, squared=False)
print(final_rmse)
```

```
Random Forest Randomized
49112.12167712042
```