School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI (ET0732)

# Topic THREE

# Launching into Machine Learning

**Preparation:**

1. Download **launching_into_ml.zip** from BrightSpace and save it into:

   **C:\users\your own folder.**

2. Extract launching_into_ml.zip

**Lab 3_1: Get to know your Data – Improve Data Quality**

Lab Intro: Lab intro: Improve the quality of your data | Google Cloud Skills Boost

Lab Demo Video: Lab Demo: Improve the quality of your data | Google Cloud Skills Boost

Lab Link: Improving Data Quality | Google Cloud Skills Boost

**Alternative Lab Instructions (on local Jupyter Notebook):**

1. In the notebook interface, navigate to **launching_into_ml > labs**, and open **improve_data_quality.ipynb**.
2. In the notebook interface, click **Edit > Clear All Outputs**.
3. Carefully read through the notebook instructions and **fill in lines marked with #TODO** where you need to complete the code as needed.

**Note:** Tips

- To run the current cell, click the cell and press **SHIFT+ENTER**. Other cell commands are listed in the notebook UI under **Run**.
- Hints may also be provided for the tasks to guide you along.
- If you need more help, look at the complete solution by navigating **launching_into_ml > solutions**, and open **improve_data_quality.ipynb**.

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI (ET0732)

**Lab 3_2: Introduction to Linear Regression**

Lab Intro: Lab intro: Introduction to linear regression | Google Cloud Skills Boost

Lab Demo: Lab Demo: Intro to Linear Regression | Google Cloud Skills Boost

Lab Link: Introduction to Linear Regression | Google Cloud Skills Boost

**Alternative Lab Instructions (on local Jupyter Notebook):**

1. In the notebook interface, navigate to **launching_into_ml > labs** and open **intro_linear_regression.ipynb**.
2. In the notebook interface, on the **Edit** menu, click **Clear All Outputs**.
3. Carefully read through the notebook instructions and **fill in lines marked with #TODO** where you need to complete the code as needed.

Tip: To run the current cell, click the cell and press SHIFT+ENTER. Other cell commands are listed in the notebook UI under **Run**.

- Hints may also be provided for the tasks to guide you along.
- If you need more help, look at the complete solution by navigating to **launching_into_ml > solutions** and opening **intro_linear_regression.ipynb**.

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

**Lecture Lab 3_1: TensorFlow Playground:**

Lecture lab: Introducing the TensorFlow Playground | Google Cloud Skills Boost

A Neural Network Playground (No.1)

**Lecture Lab 3_2: TensorFlow Playground Advanced:**

Lecture lab: TensorFlow Playground - Advanced | Google Cloud Skills Boost

A Neural Network Playground (No.2)

**Lecture Lab 3: Practicing with neural network**

Lecture lab: Practicing with neural networks | Google Cloud Skills Boost

A Neural Network Playground (No.3)

School of Electrical and Electronic Engineering
Singapore Polytechnic
Machine Learning & AI (ET0732)

# ML Applications Case Study

# 1 Survival Prediction of the Titanic

## 1.1 Introduction

### 1.1.1 About This Lab

This experiment is to predict whether passengers on the Titanic can survive based on the Titanic datasets.

### 1.1.2 Objectives

Upon completion of this task, you will be able to:

- Use the Titanic datasets open to the Internet as the model input data.
- Build, train, and evaluate machine learning models
- Understand the overall process of building a machine learning model.

### 1.1.3 Datasets and Frameworks

This experiment is based on **train.csv** and **test.csv**. **test.csv** contains the result about whether the passengers can survive. This dataset has no target, that is, no result, and can be used as a real-world dataset. Involved parameters are as follows:

- **PassengerId**: passenger ID
- **Pclass**: cabin class (class 1/2/3)
- **Name**: passenger name
- **Sex**: gender
- **Age**: age
- **SibSp**: number of siblings/number of spouses
- **Parch**: number of parents/number of children
- **Ticket**: ticket No.
- **Fare**: ticket price
- **Cabin**: cabin No.
- **Embarked**: port of boarding

# 1.2 Procedure

## 1.2.1 Importing Related Libraries

```
import pandas as pd
import numpy as np
import random as rnd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
```

## 1.2.2 Importing Datasets

Step 1 Read data.

```
train_df = pd.read_csv('./train.csv')
test_df = pd.read_csv('./test.csv')
combine = [train_df, test_df]
```

Step 2 View data.

```
print(train_df.columns.values)
```

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

The first five rows of data are displayed.

```
train_df.head()
```

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22.0 | 1 | 0 | A/5 21171 | 7.2500 | NaN | S |
| 1 | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38.0 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 2 | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26.0 | 0 | 0 | STON/O2. 3101282 | 7.9250 | NaN | S |
| 3 | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35.0 | 1 | 0 | 113803 | 53.1000 | C123 | S |
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35.0 | 0 | 0 | 373450 | 8.0500 | NaN | S |

The last five rows of data are displayed.

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **886** | 887 | 0 | 2 | Montvila, Rev. Juozas | male | 27.0 | 0 | 0 | 211536 | 13.00 | NaN | S |
| **887** | 888 | 1 | 1 | Graham, Miss. Margaret Edith | female | 19.0 | 0 | 0 | 112053 | 30.00 | B42 | S |
| **888** | 889 | 0 | 3 | Johnston, Miss. Catherine Helen "Carrie" | female | NaN | 1 | 2 | W./C. 6607 | 23.45 | NaN | S |
| **889** | 890 | 1 | 1 | Behr, Mr. Karl Howell | male | 26.0 | 0 | 0 | 111369 | 30.00 | C148 | C |
| **890** | 891 | 0 | 3 | Dooley, Mr. Patrick | male | 32.0 | 0 | 0 | 370376 | 7.75 | NaN | Q |

The data overview helps check whether some data is missing and what the data type is.

```
train_df.info()
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

The related numeric-type information of the data helps check the average value and other statistics.

```
train_df.describe()
```

| | PassengerId | Survived | Pclass | Age | SibSp | Parch | Fare |
|---|---|---|---|---|---|---|---|
| **count** | 891.000000 | 891.000000 | 891.000000 | 714.000000 | 891.000000 | 891.000000 | 891.000000 |
| **mean** | 446.000000 | 0.383838 | 2.308642 | 29.699118 | 0.523008 | 0.381594 | 32.204208 |
| **std** | 257.353842 | 0.486592 | 0.836071 | 14.526497 | 1.102743 | 0.806057 | 49.693429 |
| **min** | 1.000000 | 0.000000 | 1.000000 | 0.420000 | 0.000000 | 0.000000 | 0.000000 |
| **25%** | 223.500000 | 0.000000 | 2.000000 | 20.125000 | 0.000000 | 0.000000 | 7.910400 |
| **50%** | 446.000000 | 0.000000 | 3.000000 | 28.000000 | 0.000000 | 0.000000 | 14.454200 |
| **75%** | 668.500000 | 1.000000 | 3.000000 | 38.000000 | 1.000000 | 0.000000 | 31.000000 |
| **max** | 891.000000 | 1.000000 | 3.000000 | 80.000000 | 8.000000 | 6.000000 | 512.329200 |

The character-type information helps check the number of types, the type with the maximum value, and the frequency.

```
train_df.describe(include=['O'])
```

| | Name | Sex | Ticket | Cabin | Embarked |
|---|---|---|---|---|---|
| **count** | 891 | 891 | 891 | 204 | 889 |
| **unique** | 891 | 2 | 681 | 147 | 3 |
| **top** | Vovk, Mr. Janko | male | CA. 2343 | B96 B98 | S |
| **freq** | 1 | 577 | 7 | 4 | 644 |

Step 3 Check the survival probability corresponding to each feature based on statistics.

```
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(by='Survived',
ascending=False)
```

| | Pclass | Survived |
|---|---|---|
| **0** | 1 | 0.629630 |
| **1** | 2 | 0.472826 |
| **2** | 3 | 0.242363 |

The intuitive data shows that passengers in class 1 cabins are more likely to survive.

```
train_df[["SibSp", "Survived"]].groupby(['SibSp'], as_index=False).mean().sort_values(by='Survived',
ascending=False)
```

| | SibSp | Survived |
|---|---|---|
| **1** | 1 | 0.535885 |
| **2** | 2 | 0.464286 |
| **0** | 0 | 0.345395 |
| **3** | 3 | 0.250000 |
| **4** | 4 | 0.166667 |
| **5** | 5 | 0.000000 |
| **6** | 8 | 0.000000 |

The survival probability can be directly determined by the number of siblings.

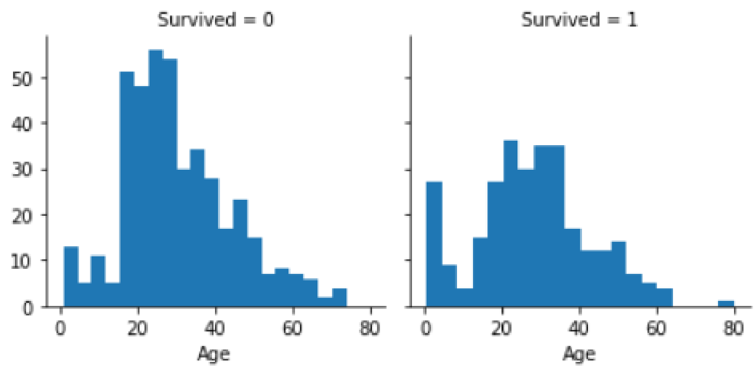
```
train_df[["Sex", "Survived"]].groupby(['Sex'], as_index=False).mean().sort_values(by='Survived', ascending=False)
```

| | Sex | Survived |
|---|---|---|
| **0** | female | 0.742038 |
| **1** | male | 0.188908 |

When the survival probability is determined by gender, an obvious imbalance occurs.

```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

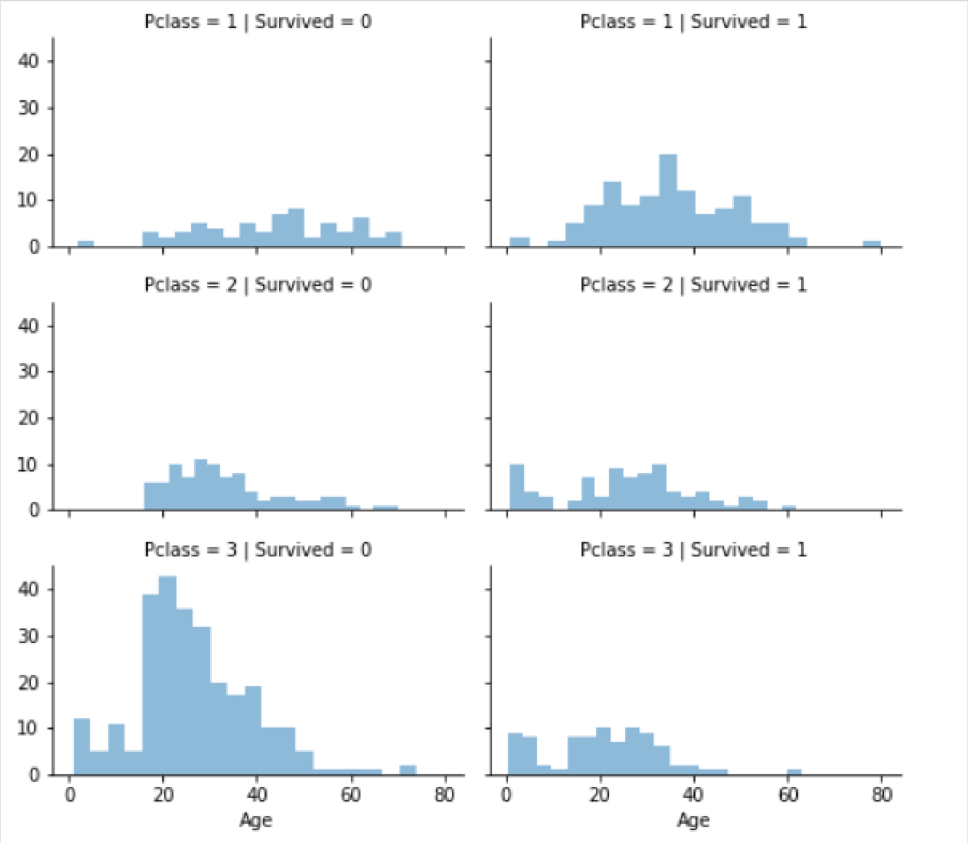
```
<seaborn.axisgrid.FacetGrid at 0x1a1df099a88>
```




As shown in the preceding figure, most young passengers died.

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass',  aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend();
```

The following figure shows the survival probability determined based on the cabin and age.




## 1.2.3 Preprocessing Data

As the datasets have missing values, combine the datasets, and fill the missing values with data.

Step 1 Combine the datasets.

```
data=pd.concat([train_df,test_df],ignore_index=True)
```

Step 2 Check for missing values.

```
data.isnull().sum()
```

```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
dtype: int64
```

Step 3 Fill the missing values with data.

Process the datasets by using different methods as required. For example, fill the **Fare** and **Embarked** parameters having few missing values with the mode.

```
data['Embarked'].fillna(str(data['Embarked'].mode()[0]),inplace=True)
data['Fare'].fillna(int(data['Fare'].mode()[0]),inplace=True)
```

Use the average age value.

```
data['Age'].fillna(data['Age'].mean(),inplace=True)
```

Delete less significant data. Before this, assign a value to **Target** first.

```
Target=data['Survived']
data=data.drop(['Cabin','Name','Ticket','Survived'],axis=1)
```

Check whether missing values still exist.

```
data.isnull().sum()
```

Step 4 Convert data.

Convert some character-type data into numeric-type data for model input. To do so, check the number of types first.

```
data['Sex'].value_counts()
```

```
male      843
female    466
Name: Sex, dtype: int64
```

Use the search function to obtain each character-type value and replace it with a numeric-type value.

```
data['Sex']=data['Sex'].replace(['male','female'],[0,1])
data['Embarked']=data['Embarked'].replace(['S','C','Q'],[0,1,2])
```

**test.csv** cannot be used as a training test set as it does not contain **Target**. **train.csv** contains 891 pieces of data (with **Target**), which need to be extracted.

```
X=data[:891]
y=Target[:891]
```

## 1.2.4 Building a Model

This section describes how to build a model. To build a model, split the training set and test set.

Step 1 Split the dataset.

```
from sklearn.model_selection import train_test_split
train_x,test_x,train_y,test_y=train_test_split(X,y)
```

Step 2 Train a model.

The logistic regression algorithm, random forest algorithm, and AdaBoost are used for training.

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import ensemble
model1=LogisticRegression()
model1.fit(X,y)
print('logR',model1.score(X,y))
model2=RandomForestClassifier()
model2.fit(X,y)
print('RFC',mode2l.score(X,y))
model3=ensemble.AdaBoostClassifier()
model3.fit(X,y)
print('AdaBoost',model3.score(X,y))
```

```
logR 0.7488789237668162
RFC 0.8071748878923767
AdaBoost 0.7757847533632287
```

As shown above, the random forest algorithm has a good effect.

Step 3 Predict data.

```
model3.predict(data[891:])
```

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

# 2 California Housing Price Forecast

## 2.1 Introduction

### 2.1.1 About This Lab

This experiment uses a dataset with a small sample quantity. The dataset includes the open-source California housing price data provided by scikit-learn. The California housing price forecast project is a simple regression model. By using this model, you can understand the basic usage and data processing methods of the machine learning library **sklearn**.

### 2.1.2 Objectives

Upon completion of this task, you will be able to:

- Use the California housing price dataset open to the Internet as the model input data.
- Build, train, and evaluate machine learning models
- Understand the overall process of building a machine learning model.
- Master the application of machine learning model training, grid search, and evaluation indicators.
- Master the application of related APIs.

### 2.1.3 Experiment Dataset and Framework

This experiment is based on the California housing price dataset, which contains 20640 samples with 8 features. Each data record contains detailed information about the house and its surroundings. LSTAT: % lower status of the population

The target is to obtain the median value of owner-occupied homes in the unit of $1000.

The **sklearn** framework is used to provide the Boston housing price data and functions such as dataset splitting, standardization, and evaluation, and integrate various common machine learning algorithms. In addition, XGBoost optimized from gradient boosted decision tree (GBDT) is used as the integral algorithm.

## 2.2 Procedure

### 2.2.1 Introducing the Dependency

Code:

```
#Prevent unnecessary warnings.
import warnings
warnings.filterwarnings("ignore")
```

```
#Introduce the basic package of data science.
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import pandas as pd
import scipy.stats as st
import seaborn as sns
##Set attributes to prevent garbled characters in Chinese.
mpl.rcParams['font.sans-serif'] = [u'SimHei']
mpl.rcParams['axes.unicode_minus'] = False

#Introduce machine learning, preprocessing, model selection, and evaluation indicators.
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score

#Import the Boston dataset used this time.
from sklearn.datasets import fetch_california_housing

#Introduce algorithms.
from sklearn.linear_model import RidgeCV, LassoCV, LinearRegression, ElasticNet
#Compared with SVC, it is the regression form of SVM.
from sklearn.svm import SVR
#Integrate algorithms.
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
```

### 2.2.2 Loading the Dataset, Viewing Data Attributes, and Visualizing Data

Step 1 Load the California housing price dataset and display related attributes.

Code:

```
#Load the Boston house price dataset.
cali = fetch_california_housing()

#x features, and y labels.
x = cali.data
y = cali.target

#Display related attributes.
print('Feature column name')
print(cali.feature_names)
print("Sample data volume: %d, number of features: %d"% x.shape)
print("Target sample data volume: %d"% y.shape[0])
```

Output:

```
Feature column name
```

['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']

Sample data volume: 20640, number of features: 8

Target sample data volume: 20640

Step 2 Convert the data into the data frame format

Code:

```
x = pd.DataFrame(cali.data, columns=cali.feature_names)
x.head()
```

Output:

| | 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 |

**Figure 2-1 Information about the first five samples**

Step 3 Visualize the label distribution.

Code:

```
sns.distplot(tuple(y), kde=False, fit=st.norm)
```

Output:

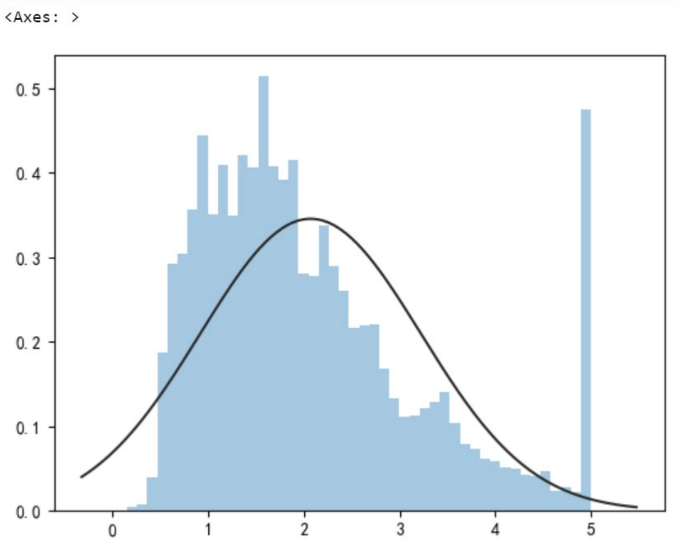



**Figure 2-2 Target data distribution**

## 2.2.3 Splitting and Preprocessing the Dataset

Code:

```
#Segment the data.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=28)
#Standardize the dataset.
ss = StandardScaler()
x_train = ss.fit_transform(x_train)
x_test = ss.transform(x_test)
x_train[0:100]
```

Output:

```
array([[ 2.28655275e+00, -1.00374733e+00,  7.31500047e-01,
        -2.81832382e-01,  7.21872633e-02, -1.31009118e-03,
         7.44083934e-01, -1.15694555e+00],
       [ 1.57815957e+00,  1.06417939e+00,  3.90085610e-01,
        -1.14864509e-01, -6.96366263e-01, -1.93935864e-02,
        -7.89623345e-01,  7.17671737e-01],
........
```

## 2.2.4 Performing Modeling on the Dataset by Using Various Regression Models

Code:

```
#Set the model name.
names = ['LinerRegression',
     'Ridge',
     'Lasso',
     'Random Forrest',
     'GBDT',
     'ElasticNet'
     ]
#Define the model.
# cv is the cross-validation idea here.
models = [LinearRegression(),
      RidgeCV(alphas=(0.001,0.1,1),cv=3),
      LassoCV(alphas=(0.001,0.1,1),cv=5),
      RandomForestRegressor(n_estimators=10),
      GradientBoostingRegressor(n_estimators=30),
      ElasticNet(alpha=0.001,max_iter=10000)]

# Output the R2 scores of all regression models.
#Define the R2 scoring function.
def R2(model,x_train, x_test, y_train, y_test):
     model_fitted = model.fit(x_train,y_train)
     y_pred = model_fitted.predict(x_test)
     score = r2_score(y_test, y_pred)
     return score

#Traverse all models to score.
for name,model in zip(names,models):
     score = R2(model,x_train, x_test, y_train, y_test)
     print("{}: {:.6f}, {:.4f}".format(name,score.mean(),score.std()))
```

Output:

```
LinerRegression: 0.625378, 0.0000
Ridge: 0.625379, 0.0000
Lasso: 0.625060, 0.0000
Random Forrest: 0.795704, 0.0000
GBDT: 0.716641, 0.0000
ElasticNet: 0.625223, 0.0000
```

## 2.2.5 Adjusting Grid Search Hyperparameters

Step 1 Build a model.

Code:

```
'''
  'kernel': kernel function
  'C': SVR regularization factor
  'gamma': 'rbf', 'poly' and 'sigmoid' kernel function coefficient, which affects the model performance
'''
parameters = {
   'kernel': ['linear', 'rbf'],
   'C': [0.1, 0.5,0.9,1,5],
   'gamma': [0.001,0.01,0.1,1]
}

#Use grid search and perform cross validation.
model = GridSearchCV(SVR(), param_grid=parameters, cv=3)
model.fit(x_train, y_train)
```

Output:

```
GridSearchCV(cv=3, estimator=SVR(),
         param_grid={'C': [0.1, 0.5, 0.9, 1, 5],
                  'gamma': [0.001, 0.01, 0.1, 1],
                  'kernel': ['linear', 'rbf']})
```

Step 2 Obtain the optimal parameters.

Code:

```
print("Optimal parameter list:", model.best_params_)
print("Optimal model:", model.best_estimator_)
print("Optimal R2 value:", model.best_score_)
```

Output:

```
Optimal parameter list: {'C': 5, 'gamma': 1, 'kernel': 'rbf'}
Optimal model: SVR(C=5, gamma=1)
```

```
Optimal R2 value: 0.7515772615928471
```

Step 3 Visualize the output.

Code:

```
##Perform visualization.
ln_x_test = range(len(x_test))
y_predict = model.predict(x_test)

#Set the canvas.
plt.figure(figsize=(16,8), facecolor='w')
#Draw with a red solid line.
plt.plot (ln_x_test, y_test, 'r-', lw=2, label=u'Value')
#Draw with a green solid line.
plt.plot (ln_x_test, y_predict, 'g-', lw = 3, label=u'Estimated value of the SVR algorithm, $R^2$=%.3f' %
(model.best_score_))
#Display in a diagram.
plt.legend(loc ='upper left')
plt.grid(True)
plt.title(u"Boston Housing Price Forecast (SVM)")
plt.xlim(0, 101)
plt.show()
```

Output:

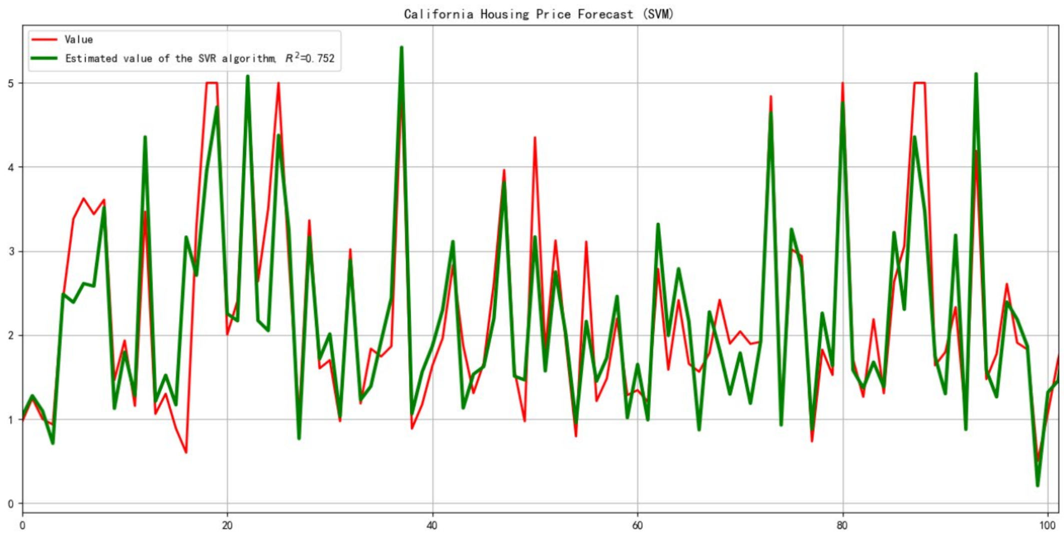



**Figure 2-3 Visualized result**

**---- End**