Open in Colab

Final Task

- 姓名: 黄长鑫
- 学号: U201915574
- 一个人一组
- 思路
  - 第一部分前四题按题目要求输出。
  - 在第二部分，首先对数据进行如下处理
    - 增加特征
    - 挑选特征
    - 去除离群点
    - 调整变量（对数处理）
  - 依次使用线性模型，随机森林等其他模型和自动机器学习模型三种模型进行训练与预测，选择预测性能最好的模型
  - 分析模型过拟合与欠拟合
- Github地址: https://github.com/For-Chance/CourseOfConstructionInformation/blob/master/FinalTask/solution.ipynb

> 因为本作业使用了较多模块，对模块的版本依赖度比较高，故推荐在colab上运行。

In [1]:
```
%%capture
# If you are not running on colab, comment out the next two lines
!apt install subversion
!svn checkout https://github.com/For-Chance/CourseOfConstructionInformation/trunk/FinalTask/dat
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
import random
random.seed(1)
%matplotlib inline
```

1 Part 1 —— Data Analysis

In [3]:
```
trans = pd.read_csv('./data/transactions.csv')
agency = pd.read_csv('./data/agency.csv')
agents = pd.read_csv('./data/agents.csv')

df = pd.merge(agency, agents)
df = pd.merge(df, trans)
df.head()
```

Out[3]:

	AgencyId	Name	AgentId	FirstName	LastName	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	lon
0	0	Other	0	Other	Other	2012.667	20.4	2469.64500	4	24.96108	121
1	0	Other	0	Other	Other	2013.167	16.2	289.32480	5	24.98203	121

	AgencyId	Name	AgentId	FirstName	LastName	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	lon
2	0	Other	0	Other	Other	2012.667	29.4	4510.35900	1	24.94925	121
3	0	Other	0	Other	Other	2012.833	31.7	1160.63200	0	24.94968	121
4	0	Other	0	Other	Other	2013.583	6.6	90.45606	9	24.97433	121

1.1 Task 1 —— 5 points

> **Question**: Prepare a table that presents how many transactions were conducted by each agency. The results should be sorted by the number of transactions - descending. Each row of your table shall contain at least a number of transactions and the name of the agency.

In [4]:
```
# Aggregation of Name properties
AgcyTrans = df.groupby('Name').agg({'Name': 'count'})

# Rename
AgcyTrans.index.name = 'Agency'
AgcyTrans = AgcyTrans.rename(columns = {'Name': 'Number of Transcations'})

# Sort
AgcyTrans = AgcyTrans.sort_values('Number of Transcations', ascending=False)

AgcyTrans
```

Out[4]:

Agency	Number of Transcations
Your Estate	225
Lovely Housing	150
Other	39

1.2 Task 2 —— 5 points

> **Question**: Prepare a table that presents the mean house price of the unit area for each agent (mean value of column Y for each agent). The results shall be sorted by the mean price - descending.

In [5]:
```
# Aggregation of Name properties
AgtPrice = df.groupby(['AgentId']).agg({'Y house price of unit area': 'mean'})

# Rename
AgtPrice = AgtPrice.rename(columns = {
    'Y house price of unit area': 'Mean House Price of the Unit Area'
    })

# Sort
AgtPrice = AgtPrice.sort_values('Mean House Price of the Unit Area', ascending=False)

AgtPrice
```

Out[5]:

AgentId	Mean House Price of the Unit Area
5	41.797727
1	38.694595
4	38.176316
3	37.961290
2	36.242045
0	35.902564

1.3 Task 3 —— 5 points

> **Question**: Prepare a table that presents the mean house price of the unit area for each agency and for each year. The results shall be sorted by the year (ascending) and then by the name of the agency. Hint: You may want to add a new column with year only, to solve this task.

In [6]:

```
AgtYearPrice = df[['X1 transaction date','Name','Y house price of unit area']].copy()
AgtYearPrice['X1 transaction date'] = AgtYearPrice['X1 transaction date'].astype(int)

# Rename
AgtYearPrice = AgtYearPrice.rename(columns={
    'X1 transaction date': 'Year',
    'Name': 'Name of Agency',
    'Y house price of unit area': 'Mean House Price of Unit Area'})

# Aggregation of Year properties
AgtYearPrice = AgtYearPrice.groupby(['Year','Name of Agency']).agg({
    'Mean House Price of Unit Area': 'mean'
    })

# Sort
AgtYearPrice = AgtYearPrice.sort_values(['Year', 'Name of Agency'])

AgtYearPrice
```

Out[6]:

Year	Name of Agency	Mean House Price of Unit Area
2012	Lovely Housing	38.217391
	Other	32.154545
	Your Estate	35.691304
2013	Lovely Housing	38.526923
	Other	37.375000
	Your Estate	39.077564

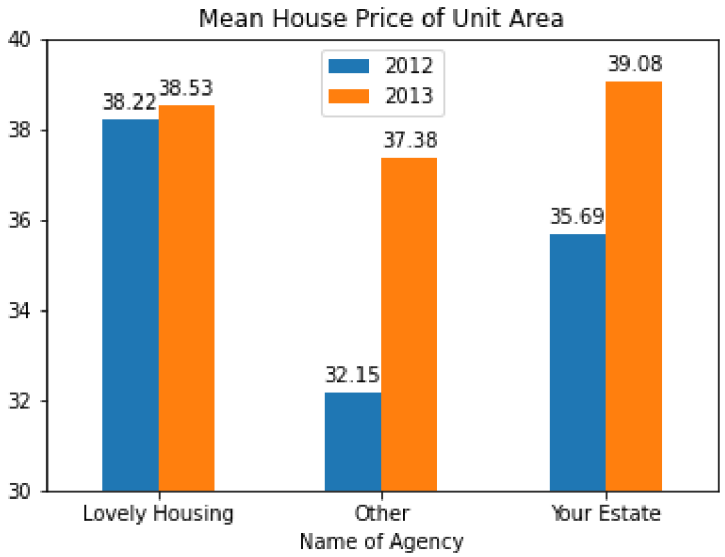
1.4 Task 4 —— 5 points

> **Question**: Prepare a chart that presents the results of task 3.

In [7]:

```
year2012 = AgtYearPrice.loc[2012,'Mean House Price of Unit Area']
year2013 = AgtYearPrice.loc[2013,'Mean House Price of Unit Area']
```

```
index = AgtYearPrice.loc[2012].index.tolist()
pd.DataFrame({'2012': year2012, '2013': year2013}, index=index).plot.bar(rot=0).grid(False)
plt.title('Mean House Price of Unit Area')
plt.xlabel('Name of Agency')
plt.subplots_adjust(bottom=0.1)
ymin=30
ymax=40
plt.ylim(ymin,ymax)
plt.legend()

def add_value_label(x_list,y_list,isleft):
  for i in range(0, len(x_list)):
    label = "{:.2f}".format(y_list[i])
    plt.annotate(label, (i,y_list[i]),textcoords="offset points",xytext=(-isleft*28.5,5))

add_value_label(index,year2012,1)
add_value_label(index,year2013,0)
```



2 Part 2 —— Machine learning

> **Question**: For this part, your goal is to prepare, train and test the Machine Learning model that can estimate the house price of unit area.

In [8]:

```
trans = pd.read_csv('./data/transactions.csv')
agency = pd.read_csv('./data/agency.csv')
agents = pd.read_csv('./data/agents.csv')

# Merge tables
df = pd.merge(agency, agents)
df = pd.merge(df, trans)

# Drop meaningless column
df = df.drop(['Name','FirstName','LastName'], axis = 1)

# Rename
df.rename(columns = {
    'X1 transaction date': 'X1',
    'X2 house age': 'X2',
    'X3 distance to the nearest MRT station': 'X3',
    'X4 number of convenience stores': 'X4',
    'X5 latitude': 'X5',
    'X6 longitude': 'X6',
    'Y house price of unit area':'Y'}, inplace=True)

source = df.copy()
df.head()
```

Out[8]:

	AgencyId	AgentId	X1	X2	X3	X4	X5	X6	Y
0	0	0	2012.667	20.4	2469.64500	4	24.96108	121.51046	23.8
1	0	0	2013.167	16.2	289.32480	5	24.98203	121.54348	46.2
2	0	0	2012.667	29.4	4510.35900	1	24.94925	121.49542	13.2
3	0	0	2012.833	31.7	1160.63200	0	24.94968	121.53009	13.7
4	0	0	2013.583	6.6	90.45606	9	24.97433	121.54310	59.0

2.1 Task 5 —— 10 points

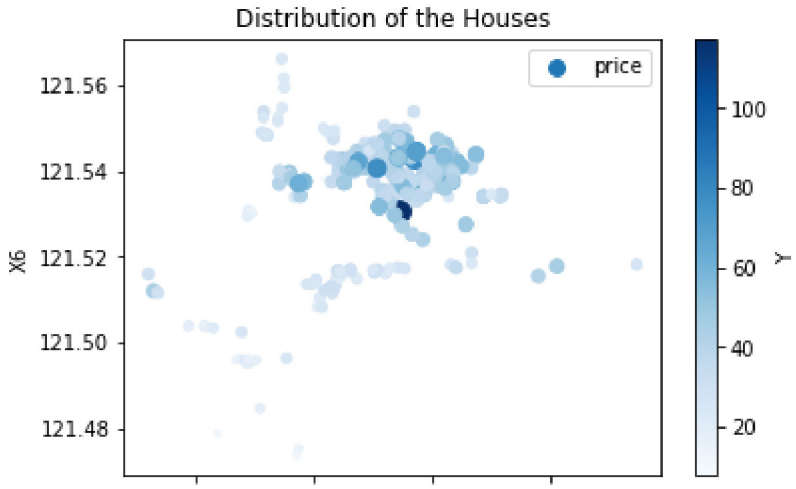
> **Question**: Select the features and labels (X and y) from your dataset. Make sure that your selection is reasonable. Split the data into two datasets (training and testing).

2.1.1 Add Features

The distribution of the houses according to $X5$ (latitude) and $X6$ (longitude) is shown below.

In [9]:

```
df.plot(
    kind='scatter',x='X5',y='X6',alpha=1,s=df['Y'],
    c='Y', label='price', colormap='Blues',colorbar=True)
plt.title("Distribution of the Houses");
```



> As shown above, the house price is clearly related to the distance of the house from the city center. Therefore, to enhance the correlation between features and house prices and to reduce the multicollinearity between latitude and longitude, we use the following formula to calculate the downtown location. and add $X7$ as the distance of the house from the city center according to the city center location.
>
> $$center_x = \frac{\sum x_i P}{\sum P}$$
> $$center_y = \frac{\sum y_i P}{\sum P}$$
>
> where (x_i, y_i) represents the location information represented by the latitude and longitude, and P is the house price.

In [10]:

```
# Calculate downtown location
center_x = sum(df['X5'] * df['Y']) / df['Y'].sum()
center_y = sum(df['X6'] * df['Y']) / df['Y'].sum()
# One unit of latitude and longitude is approximately 100 km
```

```
df['X7'] = ((df['X5']-center_x)**2+(df['X6']-center_y)**2)**0.5 * 100000
df = df.drop(columns=['X5', 'X6'])
print(f'downtown loation is ({center_x:.4f}, {center_y:.4f})')
df.head()
```

```
downtown loation is (24.9715, 121.5362)
```

Out[10]:

	AgencyId	AgentId	X1	X2	X3	X4	Y	X7
0	0	0	2012.667	20.4	2469.64500	4	23.8	2778.053397
1	0	0	2013.167	16.2	289.32480	5	46.2	1282.246917
2	0	0	2012.667	29.4	4510.35900	1	13.2	4646.003385
3	0	0	2012.833	31.7	1160.63200	0	13.7	2262.259333
4	0	0	2013.583	6.6	90.45606	9	59.0	744.690602

2.1.2 Select features

In [11]:

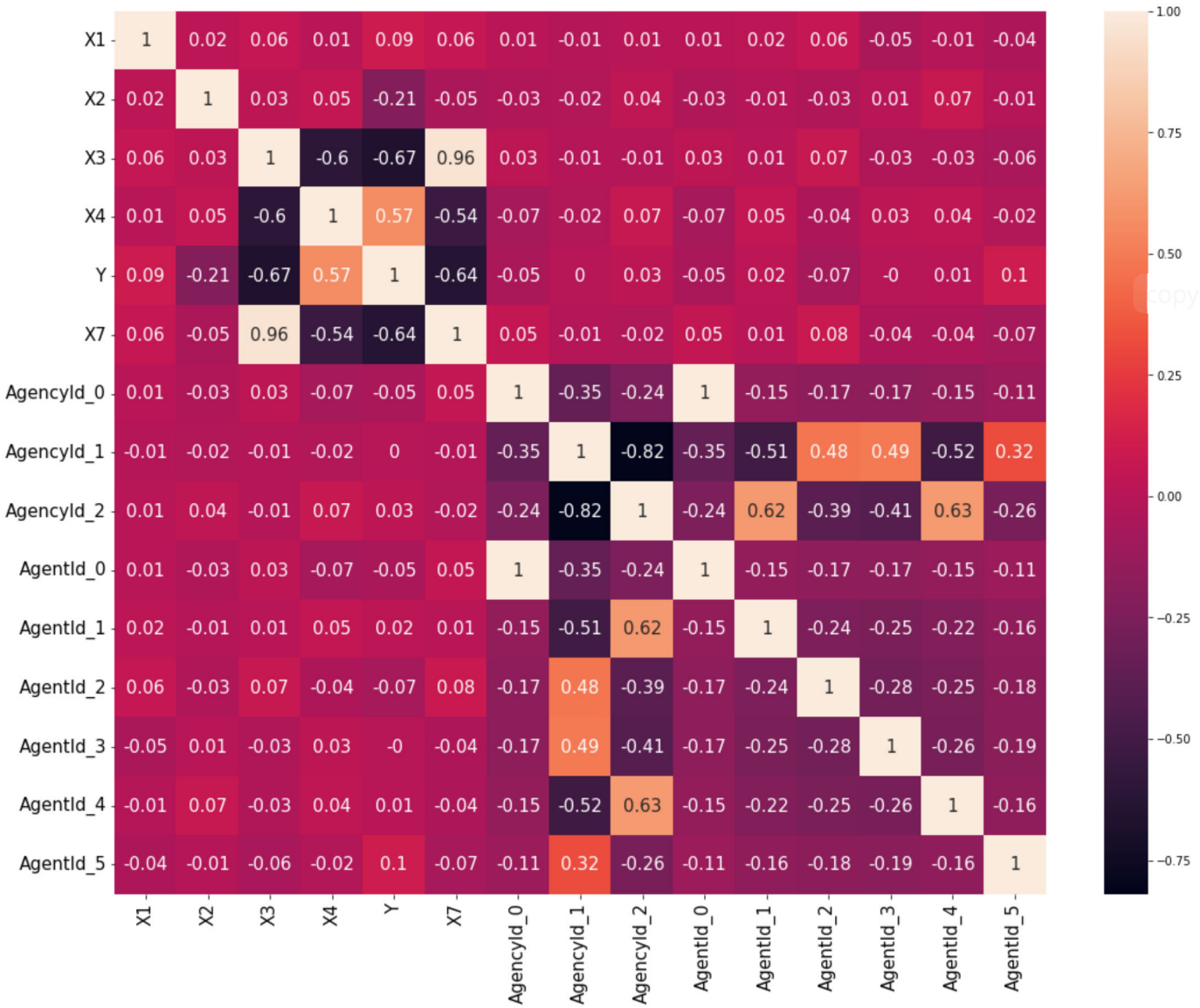
```
# One-Hot Encoding
df = pd.get_dummies(df, columns=['AgencyId','AgentId'])
df.head()
```

Out[11]:

	X1	X2	X3	X4	Y	X7	AgencyId_0	AgencyId_1	AgencyId_2	AgentId_0	AgentId_
0	2012.667	20.4	2469.64500	4	23.8	2778.053397	1	0	0	1	
1	2013.167	16.2	289.32480	5	46.2	1282.246917	1	0	0	1	
2	2012.667	29.4	4510.35900	1	13.2	4646.003385	1	0	0	1	
3	2012.833	31.7	1160.63200	0	13.7	2262.259333	1	0	0	1	
4	2013.583	6.6	90.45606	9	59.0	744.690602	1	0	0	1	

In [12]:

```
# Correlation coefficient matrix
import seaborn as sns
plt.subplots(figsize=(18, 14))
sns.heatmap(df.corr().round(2),annot=True,annot_kws={"fontsize":15})
plt.xticks(fontsize=15)
plt.yticks(fontsize=15);
```



According to the correlation coefficients of the target Y columns and other features, the columns with correlation coefficients greater than 0.5 are selected, and then $X5$ (latitude) and $X6$ (longitude) columns are removed, and the final selected features are $X3$,$X4$ and $X7$ columns.

In [13]:
```
df = df[['X3','X4','X7','Y']]
df.head()
```

Out[13]:

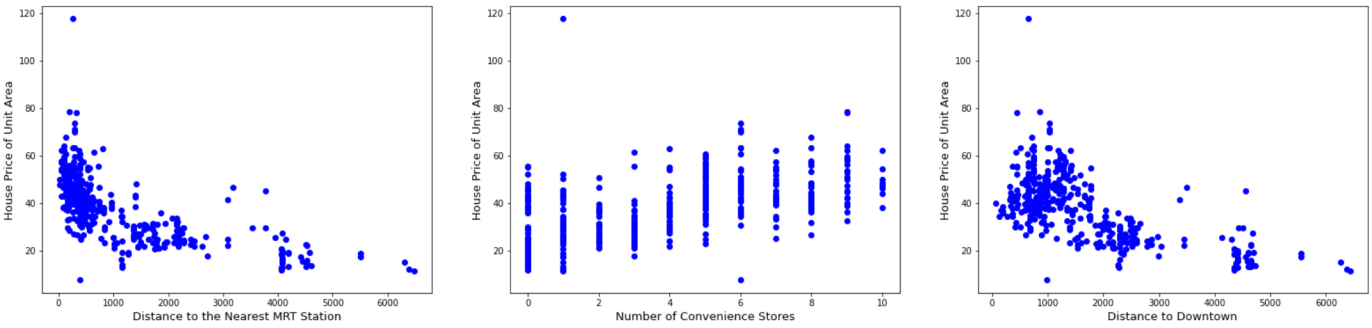
	X3	X4	X7	Y
0	2469.64500	4	2778.053397	23.8
1	289.32480	5	1282.246917	46.2
2	4510.35900	1	4646.003385	13.2
3	1160.63200	0	2262.259333	13.7
4	90.45606	9	744.690602	59.0

2.1.3 Removing outliers

In [14]:
```
# Plotting scatter plots of different characteristics with house prices
def plotScatter():
  plt.figure(figsize=(27, 6))

  plt.subplot(1, 3, 1)
  plt.scatter(x=df.X3, y=df.Y, color='b')
  plt.xlabel("Distance to the Nearest MRT Station", fontsize=13)
  plt.ylabel("House Price of Unit Area", fontsize=13)
```

```
  plt.subplot(1, 3, 2)
  plt.scatter(x=df.X4, y=df.Y, color='b')
  plt.xlabel("Number of Convenience Stores", fontsize=13)
  plt.ylabel("House Price of Unit Area", fontsize=13)

  plt.subplot(1, 3, 3)
  plt.scatter(x=df.X7, y=df.Y, color='b')
  plt.xlabel("Distance to Downtown", fontsize=13)
  plt.ylabel("House Price of Unit Area", fontsize=13)

print(f'df shape: {df.shape}')
plotScatter()
```

```
df shape: (414, 4)
```

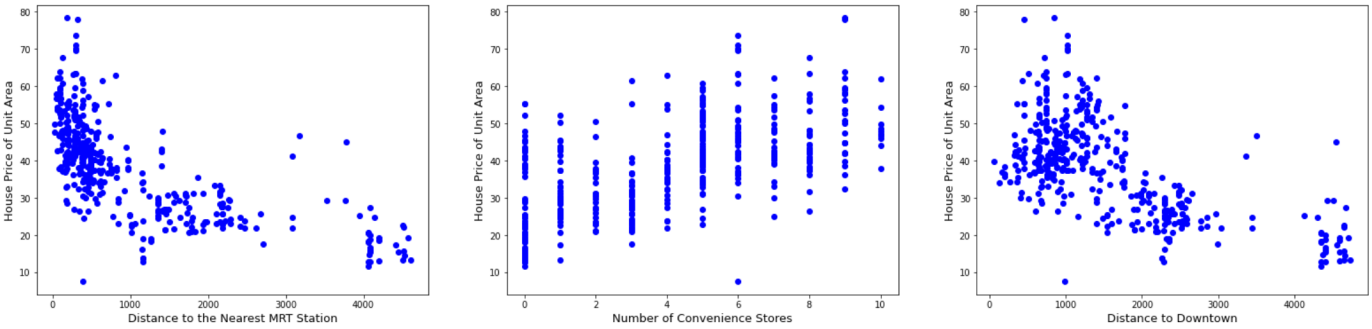


Assuming that the data are normally distributed, we can therefore remove outliers according to the 3σ principle

In [15]:

```
# Removing outliers
def remove_outliers(df, columns=[]):
  mean, std = df.mean(), df.std()
  cut_off = std * 3
  lower, upper = mean - cut_off, mean + cut_off

  for col in columns:
    df = df[(df[col] > lower[col]) & (df[col] < upper[col])]
  return df

df = remove_outliers(df, columns=['X3','X4','X7','Y'])
print(f'df shape: {df.shape}')
plotScatter()
```

```
df shape: (408, 4)
```



2.1.4 Adjusting variables

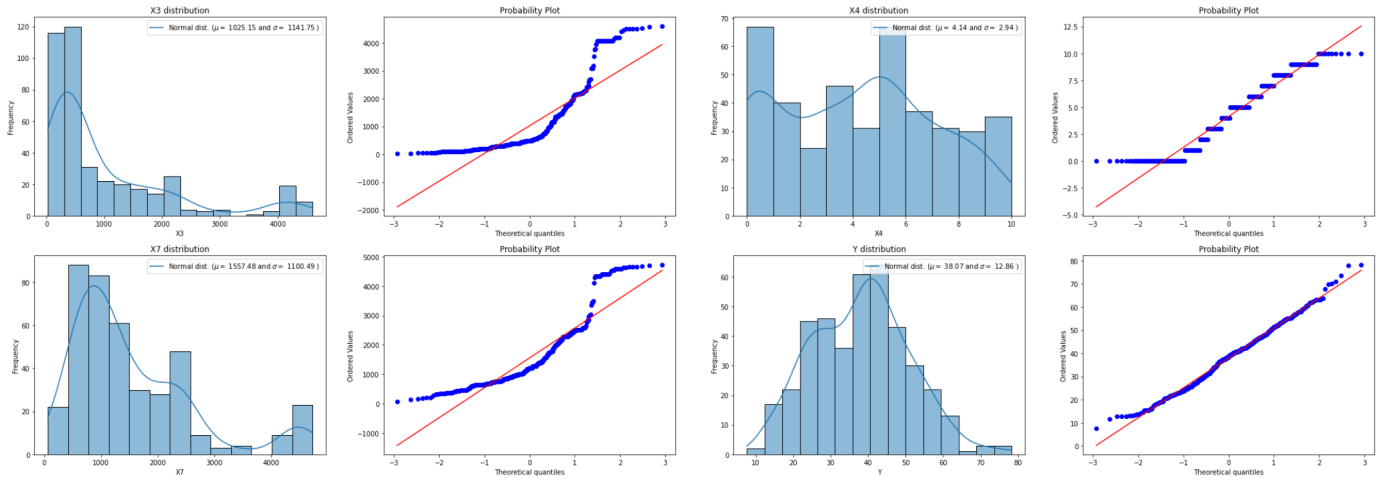
In [16]:

```
from scipy.stats import norm
from scipy import stats
```

In [17]:

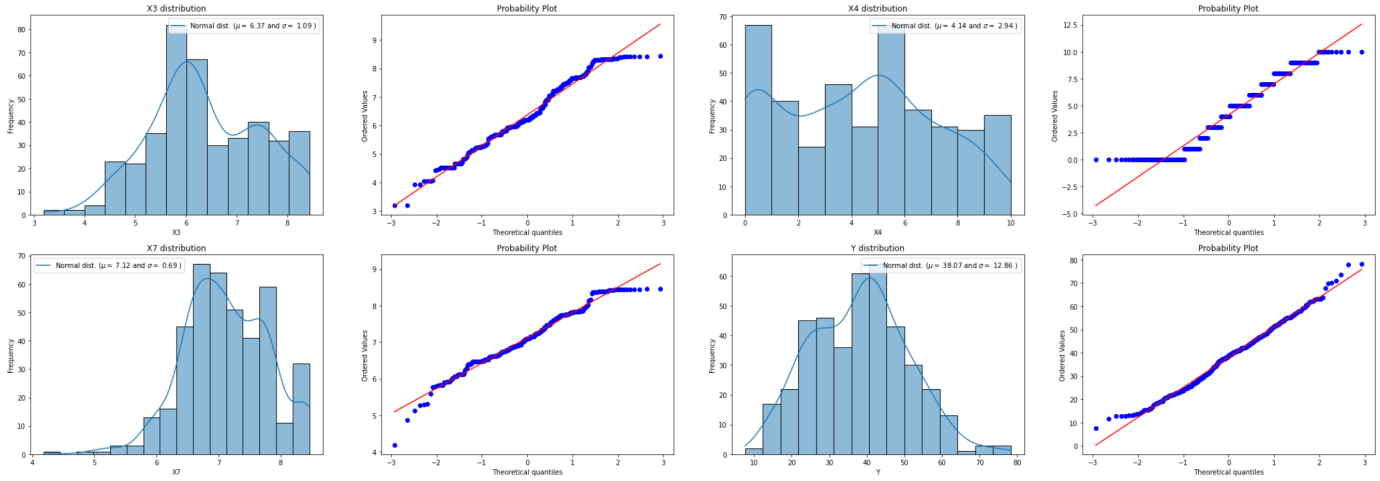
```
def plt_distribution(data):
  """Plotting histograms and probability distributions"""
  plt.figure(figsize=(36,12))

  cnt = 0
```

```
  for col in data:
    plt.subplot(2,4,2*cnt+1)
    sns.histplot(data[col], kde=True);

    (mu, sigma) = norm.fit(data[col])
    # print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

    plt.legend(
        ['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
        loc='best')
    plt.ylabel('Frequency')
    plt.title(f'{col} distribution')

    plt.subplot(2,4,2*cnt+2)
    stats.probplot(data[col], plot=plt)
    cnt += 1
plt_distribution(df)
```



According to the above graph we can see that both $X3$ and $X7$ deviate more from the normal distribution, so we correct them using the logarithmic treatment.

In [18]:
```
df['X3'] = np.log1p(df['X3'])
df['X7'] = np.log1p(df['X7'])
df.head()
plt_distribution(df)
```



2.2 Task 6 —— 10 points

> **Question**: Prepare the Machine Learning model, then train and test this model. For this task, you should provide your code in Python as well as the accuracy of the model. As an accuracy, you should report Mean Absolute Error and Mean Absolute Percentage Error. You may also report other metrics (R2, SMAPE, MSE) but this is not necessary.

Use the following metrics to evaluate the model:

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

$$MAPE = \frac{100\%}{n}\sum_{i=1}^{n}|\frac{\hat{y}_i - y_i}{y_i}|$$

$$R2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(\bar{y} - y_i)^2}$$

$$SMAPE = \frac{100\%}{n}\sum_{i=1}^{n}\frac{|\hat{y}_i - y_i|}{(|y_i| + |\hat{y}_i|)/2}$$

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

where $\hat{y}_i$ is the house price forecast, y_i the true value and $\overline{y}$ the average value. **The better the model, the smaller the values of MAE, MAPE, SMAPE and MSE, and the larger the value of R2.**

In [19]:

```
from sklearn import metrics
from sklearn.metrics import r2_score

def get_mae(y, pred):
  return metrics.mean_absolute_error(y, pred)

def get_mape(y, pred):
  return np.mean(100 * np.abs((y-pred) / y))

def get_mape_model(model, X, y):
    pred = model.predict(X)
    return np.mean(100 * np.abs((y-pred) / y))

def get_R2(y, pred):
  return r2_score(y, pred)

def get_smape(y, pred):
  return 2.0 * np.mean(np.abs(pred - y) / (np.abs(pred) + np.abs(y))) * 100

def get_mse(y, pred):
  return metrics.mean_squared_error(y, pred)

def splitData(df,random_state=None):
  """Slice the df into training and test sets according to random_state"""
  X = df.drop(columns='Y').copy()
  y = df['Y']

  X_train, X_test, y_train, y_test = train_test_split(
      X, y, test_size=0.25, random_state=random_state, shuffle=True)

  # Standardization
  scaler=StandardScaler()
  X_train = scaler.fit_transform(X_train)
  X_test = scaler.fit_transform(X_test)

  return (X_train, X_test, y_train, y_test)


def show_result_loop(df, model, loop=20, isAutoML=False):
  '''The results are fittedloop times to obtain the average of the evaluation indicators of the m
  metrics_train = [0,0,0,0,0]
  metrics_test = [0,0,0,0,0]
  for i in range(loop):
```

```
    if isAutoML:
      # split data
      df = shuffle(df,random_state=12).reset_index(drop=True)
      split_point = int(len(df)*0.75)
      train_data, test_data = df[:split_point], df[split_point:]

      train_data = TabularDataset(train_data)
      test_data = TabularDataset(test_data)
      save_path = 'best_model'
      model = TabularPredictor(label='Y', problem_type='regression', path=save_path, verbosity=0
        train_data
      )
      model = TabularPredictor.load(save_path)
      X_train = train_data.drop(columns='Y')
      X_test = test_data.drop(columns='Y')
      y_train = train_data['Y']
      y_test = test_data['Y']
    else:
      (X_train, X_test, y_train, y_test) = splitData(df,random_state=None)
      model.fit(X_train, y_train)

    pred = model.predict(X_train)
    metrics_train[0] += get_mae(y_train, pred)
    metrics_train[1] += get_mape(y_train, pred)
    metrics_train[2] += get_R2(y_train, pred)
    metrics_train[3] += get_smape(y_train, pred)
    metrics_train[4] += get_mse(y_train, pred)

    pred = model.predict(X_test)
    metrics_test[0] += get_mae(y_test, pred)
    metrics_test[1] += get_mape(y_test, pred)
    metrics_test[2] += get_R2(y_test, pred)
    metrics_test[3] += get_smape(y_test, pred)
    metrics_test[4] += get_mse(y_test, pred)
  for i in range(5):
    metrics_train[i] /= loop
    metrics_test[i] /= loop

  print("\t\tMAE\tMAPE\tR2\tSMAPE\tMSE")
  print(f"Train accurary:\t{metrics_train[0]:.2f}\t{metrics_train[1]:.2f}%\t{metrics_train[2]:.21
  print(f"Test accurary:\t{metrics_test[0]:.2f}\t{metrics_test[1]:.2f}%\t{metrics_test[2]:.2f}\t
```

Define a class to facilitate the use of grid search hyperparameters.

In [20]:

```
from sklearn.model_selection import GridSearchCV
class grid():
  def __init__(self,model):
    self.model = model

  def grid_get(self,X,y,param_grid):
    grid_search = GridSearchCV(
        self.model,param_grid,cv=5,scoring="neg_mean_squared_error")
    grid_search.fit(X,y)
    print(f'Best Params: {grid_search.best_params_}, \
    RMSE: {np.sqrt(-grid_search.best_score_):.4f}')
```

2.2.1 Linear model

> Using the linear model requires processing of the data in Task 5. Therefore, if it is executed sequentially, there will be no problem.

In [21]:

```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet, BayesianRidge
```

In [22]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

(X_train, X_test, y_train, y_test) = splitData(df,random_state=1)

print('X train shape:', X_train.shape)
print('X test shape:', X_test.shape)
print('y train shape:', y_train.shape)
print('y test shape:', y_test.shape)
```

```
X train shape: (306, 3)
X test shape: (102, 3)
y train shape: (306,)
y test shape: (102,)
```

In [23]:
```
# Grid search for optimal hyperparameter values
print("Ridge:")
param_grid = {'alpha':[35,40,45,50,55,60,65,70,80,90]}
grid(Ridge()).grid_get(X_train, y_train, param_grid)
print("Lasso:")
param_grid = {
    'alpha': [0.0004,0.0005,0.0006,0.0007,0.0008,0.0009],
    'max_iter':[10000],
    'random_state':[1]}
grid(Lasso()).grid_get(X_train, y_train, param_grid)
print("ElasticNet:")
param_grid = {
    'alpha':[0.0005,0.0008,0.004,0.005],
    'l1_ratio':[0.08,0.1,0.3,0.5,0.7],
    'max_iter':[10000],
    'random_state':[1]}
grid(ElasticNet()).grid_get(X_train, y_train, param_grid)
```

```
Ridge:
Best Params: {'alpha': 35},      RMSE: 8.8005
Lasso:
Best Params: {'alpha': 0.0009, 'max_iter': 10000, 'random_state': 1},      RMSE: 8.7729
ElasticNet:
Best Params: {'alpha': 0.005, 'l1_ratio': 0.08, 'max_iter': 10000, 'random_state': 1},      RMSE:
8.7718
```

In [24]:
```
%%time
# Obtain linear model prediction results.
# Change the random_state of the model 20 times
# to obtain the average value of the evaluation index.
model = LinearRegression()
print("LinearRegression:")
show_result_loop(df, model, loop=20)

model = Ridge(alpha=34)
print("\nRidge:")
show_result_loop(df, model, loop=20)

model = Lasso(alpha=0.004,max_iter=10000,random_state=1)
print("\nLasso:")
show_result_loop(df, model, loop=20)

model = ElasticNet(
    alpha=0.005,l1_ratio=0.08,max_iter=10000,random_state=1)
print("\nElasticNet:")
show_result_loop(df, model, loop=20)

model = BayesianRidge()
print("\nBayesianRidge:")
show_result_loop(df, model, loop=20)
```

```
LinearRegression:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.07    18.01%  0.58    16.49%  68.58
Test accurary:  6.38    19.34%  0.55    17.38%  75.28

Ridge:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.22    18.67%  0.58    16.94%  71.07
Test accurary:  6.31    18.97%  0.55    17.34%  69.62

Lasso:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.06    17.94%  0.58    16.47%  67.95
Test accurary:  6.41    19.46%  0.56    17.49%  76.91

ElasticNet:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.15    18.35%  0.57    16.72%  70.13
Test accurary:  6.15    18.48%  0.57    16.84%  70.23

BayesianRidge:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.08    18.14%  0.58    16.57%  68.94
Test accurary:  6.38    18.98%  0.54    17.31%  74.33
CPU times: user 2.26 s, sys: 15.6 ms, total: 2.27 s
Wall time: 4.01 s
```

The results of the above five types of linear models are all relatively close, which indicates that enhancing the complexity of the model does not enhance the model well. And it is thus clear that the model appears to be overfitted, and the reason for the overfitting is insufficient data.

2.2.2 Other model

> In this section, to reduce the complexity of the data and to reduce the effect of data multicollinearity, we use PCA (Principal Component Analysis) method to reduce the dimensionality of the data.

In [25]:

```
# Obtain initial data
df = source.copy()

# One-Hot Encoding
df = pd.get_dummies(df, columns=['AgencyId','AgentId'])
df.head()

# Removing outliers
df = remove_outliers(df, columns=['X2','X3','X4','X5','X6','Y'])

X_train, X_test, y_train, y_test = splitData(df, random_state=1)

print('X train shape:', X_train.shape)
print('X test shape:', X_test.shape)
print('y train shape:', y_train.shape)
print('y test shape:', y_test.shape)
```

```
X train shape: (305, 15)
X test shape: (102, 15)
y train shape: (305,)
y test shape: (102,)
```

In [26]:

```
# PCA downscaling
from sklearn.decomposition import PCA
def PCA_process(X_train,X_test,n_components=3):
  scaler=StandardScaler()
  pca_model = PCA(n_components=n_components)
  X_train = pca_model.fit_transform(X_train)
```

```
    X_test = pca_model.transform(X_test)
    return (X_train,X_test)
X_train,X_test = PCA_process(X_train,X_test)
print('X train shape:', X_train.shape)
print('X test shape:', X_test.shape)
```

```
X train shape: (305, 3)
X test shape: (102, 3)
```

In [27]:

```
# Importing Models
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
```

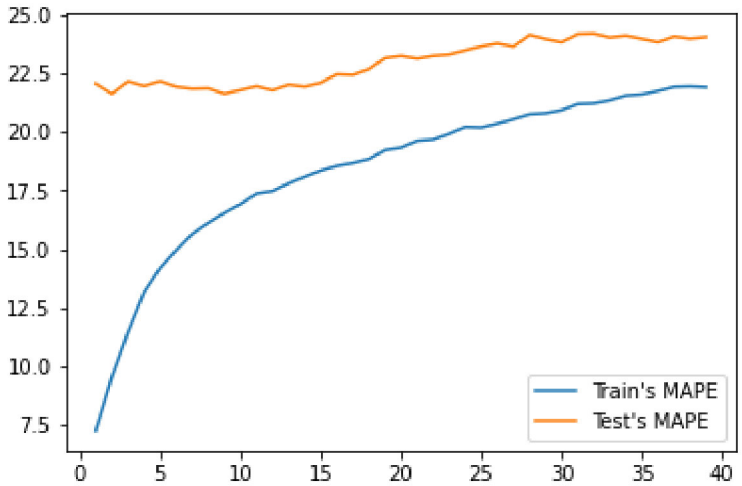
These three models are too complex for the size of the data, so in order to prevent overfitting, the following observation of the training curve is chosen for tuning.

In [28]:

```
mapes_train = []
mapes_test = []
ticks=[]

for i in np.arange(1,40):
    model = RandomForestRegressor(min_samples_leaf=i)
    model.fit(X_train, y_train)

    train_mape = get_mape_model(model, X_train, y_train)
    mapes_train.append(train_mape)
    test_mape = get_mape_model(model, X_test, y_test)
    mapes_test.append(test_mape)
    ticks.append(i)

f = plt.figure()
plt.plot(ticks,mapes_train,label='Train\'s MAPE')
plt.plot(ticks,mapes_test,label='Test\'s MAPE')
plt.legend()
```

Out[28]:

```
<matplotlib.legend.Legend at 0x7f4cb37cf7d0>
```



As above, the value of *min_samples_leaf* should be 16 in order to balance underfitting and overfitting, and other parameters are adjusted in the same way. To simplify the presentation, the model parameters in the following have been adjusted.

In [29]:

```
model = RandomForestRegressor(
        n_estimators=50,min_samples_leaf=16,max_depth=5,
        min_samples_split=60,min_weight_fraction_leaf=0.0,
        min_impurity_decrease=4)
print("RandomForestRegressor:")
show_result_loop(df, model, loop=20)
```

```
RandomForestRegressor:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 5.04    15.08%  0.70    13.89%  48.55
Test accurary:  5.80    17.78%  0.62    16.04%  63.56
```

In [30]:

```
model = XGBRegressor(
        max_depth=2,learning_rate=0.1,n_estimators=20,
        min_child_weight=10,objective ='reg:squarederror',booster="dart"
        ,subsample=0.55,colsample_bytree=0.7,reg_alpha=1.0,
        reg_lambda=1.6,base_score=0.8,random_state=1)
print("XGBRegressor:")
show_result_loop(df, model, loop=20)
```

```
XGBRegressor:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.20    15.77%  0.57    16.58%  71.44
Test accurary:  6.78    17.28%  0.49    18.23%  83.93
```

In [31]:

```
model = LGBMRegressor(
        objective='regression',n_estimators=20,learning_rate=0.1,
        max_depth=6,num_leaves=5,min_child_samples=20)
print("XGBRegressor:")
show_result_loop(df, model, loop=20)
```

```
XGBRegressor:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 4.58    14.26%  0.76    12.94%  39.07
Test accurary:  5.54    16.99%  0.65    15.33%  58.86
```

As can be seen above, these three types of more complex models do not improve the prediction performance much relative to the linear model and show more severe overfitting.

2.2.3 Auto Machine Learning

In [32]:

```
%%capture
!pip install autogluon
!pip install xgboost
!pip install lightgbm
```

In [33]:

```
# Import Module
# If the module import fails here, just restart the kernel
from autogluon.tabular import TabularDataset, TabularPredictor
```

In [34]:

```
# Obtain initial data
df = source.copy()

# One-Hot Encoding
df = pd.get_dummies(df, columns=['AgencyId','AgentId'])

# Removing outliers
df = remove_outliers(df, columns=['X2','X3','X4','X5','X6','Y'])

# split data
df = shuffle(df,random_state=12).reset_index(drop=True)
split_point = int(len(df)*0.75)
train_data, test_data = df[:split_point], df[split_point:]

train_data = TabularDataset(train_data)
test_data = TabularDataset(test_data)
X_train = train_data.drop(columns=['Y'])
y_train = train_data['Y']
X_test = test_data.drop(columns=['Y'])
y_test = test_data['Y'].reset_index(drop=True)
```

```
# PCA downscaling
(X_train, X_test) = PCA_process(X_train, X_test, 2)

# Re-merge data
X_train = pd.DataFrame(X_train, columns=['X1','X2'])
X_test = pd.DataFrame(X_test, columns=['X1','X2'])
train_data = pd.concat([X_train,y_train],axis=1)
test_data = pd.concat([X_test,y_test],axis=1)
df = pd.concat([train_data,test_data],axis=0)
print('X train shape:', train_data.shape)
print('X test shape:', test_data.shape)
print('df shape:', df.shape)
```

```
X train shape: (305, 3)
X test shape: (102, 3)
df shape: (407, 3)
```

In [35]:

```
%%time
# Demonstrate an auto-referencing process
save_path = 'best_model'
model = TabularPredictor(label='Y', problem_type='regression', path=save_path).fit(
  train_data
)
```

```
Warning: path already exists! This predictor may overwrite an existing predictor! path="best_mode
l"
Beginning AutoGluon training ...
AutoGluon will save models to "best_model/"
AutoGluon Version:  0.5.0
Python Version:     3.7.13
Operating System:   Linux
Train Data Rows:    305
Train Data Columns: 2
Label Column: Y
Preprocessing data ...
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    11892.38 MB
        Train Data (Original)  Memory Usage: 0.01 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set feature_metadata_in to ma
nually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator...
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator...
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator...
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 2 | ['X1', 'X2']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', []) : 2 | ['X1', 'X2']
        0.1s = Fit runtime
        2 features in original data used to generate 2 features in processed data.
        Train Data (Processed) Memory Usage: 0.01 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.09s ...
AutoGluon will gauge predictive performance using evaluation metric: 'root_mean_squared_error'
        This metric's sign has been flipped to adhere to being higher_is_better. The metric score
can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
Automatically generating train/validation split with holdout_frac=0.2, Train Rows: 244, Val Rows:
61
Fitting 11 L1 models ...
Fitting model: KNeighborsUnif ...
        -6.6945  = Validation score   (-root_mean_squared_error)
        0.01s    = Training   runtime
```

```
        0.12s    = Validation runtime
Fitting model: KNeighborsDist ...
        -6.8915  = Validation score   (-root_mean_squared_error)
        0.01s    = Training   runtime
        0.11s    = Validation runtime
Fitting model: LightGBMXT ...
        -6.4161  = Validation score   (-root_mean_squared_error)
        0.38s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBM ...
        -6.2829  = Validation score   (-root_mean_squared_error)
        0.32s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: RandomForestMSE ...
        -5.8657  = Validation score   (-root_mean_squared_error)
        0.62s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: CatBoost ...
        -6.3514  = Validation score   (-root_mean_squared_error)
        0.58s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: ExtraTreesMSE ...
        -6.1206  = Validation score   (-root_mean_squared_error)
        0.73s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: NeuralNetFastAI ...
No improvement since epoch 8: early stopping
        -6.5688  = Validation score   (-root_mean_squared_error)
        5.12s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: XGBoost ...
        -7.0584  = Validation score   (-root_mean_squared_error)
        0.51s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch ...
        -6.7793  = Validation score   (-root_mean_squared_error)
        2.3s     = Training   runtime
        0.02s    = Validation runtime
Fitting model: LightGBMLarge ...
        -6.5484  = Validation score   (-root_mean_squared_error)
        0.6s     = Training   runtime
        0.04s    = Validation runtime
Fitting model: WeightedEnsemble_L2 ...
        -5.7174  = Validation score   (-root_mean_squared_error)
        0.87s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 13.82s ... Best model: "WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("best_model/")
CPU times: user 11.1 s, sys: 579 ms, total: 11.6 s
Wall time: 13.9 s
```

In [36]:

```
%%time
# If there is a lot of Warning in the resulting output,
# then please restart the kernel and run it again.
show_result_loop(df,model,loop=10,isAutoML=True)
```

```
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 3.01    9.24%   0.89    8.51%   18.88
Test accurary:  5.01    14.38%  0.70    13.61%  49.66
CPU times: user 1min 32s, sys: 3.98 s, total: 1min 36s
Wall time: 1min 50s
```

It can be seen that there is a large overfitting of the model obtained by automatic machine learning. However, the prediction performance is better, so we choose it as the final model.

2.3 Task 7 – 10 points

> **Question**：Analyze your results and answer the following questions:
> • Is your model overfitting? (Yes, No, cannot say), explain your answer.
> • Is your model underfitting? (Yes, No, cannot say), explain your answer.
> For this task, you should present the answers to the questions above.

My Answer:

- Is your model overfitting?
  - Yes。
  - The very complex *WeightedEnsemble_L2* model has a MAPE of 9.24% for the training data and 14.38% for the testing data, which intuitively indicates that a relatively serious overfitting occurs. And throughout the training process, the more complex the model is from linear model to random forest model to automatic machine learning model, the more severe the overfitting becomes. Therefore, part of the overfitting is due to the model being too complex.
  - Another explainable phenomenon of the existence of overfitting is that, regardless of the above-mentioned models, the results of the models fluctuate relatively widely whenever the split between the training and test sets is randomly changed. This means that the amount of data is too small for either model to complete adequate training, which leads to overfitting. Therefore, another reason for overfitting is too few data samples.
- Is your model underfitting?
  - Cannot say.
  - We can say that the model is not underfitting. This is because the *WeightedEnsemble_L2* model is necessarily able to meet the requirements of the data in terms of matching the model to the data. And the model gradually decreases MAPE on the test data from linear model to random forest model to automatic machine learning model, but the degree of overfitting gradually increases. Therefore increasing the predictive performance of the model will come at the cost of increasing overfitting, so the model has reached the upper limit of the model that can be trained with this data size. Therefore, it can be said that the model is not underfitting.
  - We can also say that the model is underfitting. Because it can be seen from the process of PCA dimensionality reduction, the features of the data can be almost completely reduced to one column, so the features of the data are too few, and the multicollinearity between the features is serious. The feature dimensionality is too small, which results in the fitted function cannot satisfy the training data and the error is large. Therefore the model suffers from underfitting.