Open in Colab

Final Task

- 姓名: 黄长鑫
- 学号: U201915574
- 一个人一组
- 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
# 如果不是在colab上运行，请注释掉下两行
!apt install subversion
!svn checkout https://github.com/For-Chance/CourseOfConstructionInformation/trunk/FinalTask/dat
```

In [75]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
%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
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

> 题目：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]:
```
# 对Name属性聚合
AgcyTrans = df.groupby('Name').agg({'Name': 'count'})

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

# 重新排序
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

> 题目：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]:
```
# 对AgentId聚合
AgtPrice = df.groupby(['AgentId']).agg({'Y house price of unit area': 'mean'})

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

# 重新排序
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

> 题目：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)

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

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

# 重新排序
AgtYearPrice = AgtYearPrice.sort_values(['Year', 'Name of Agency'])

AgtYearPrice
```

Out[6]:

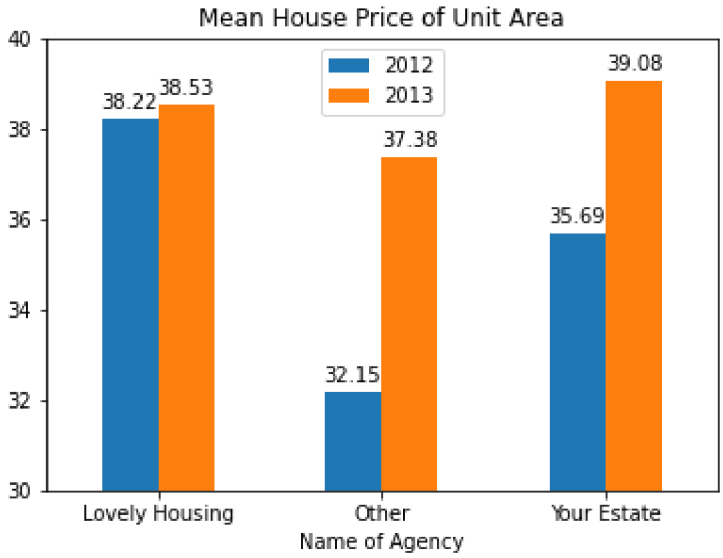
		Mean House Price of Unit Area
Year	**Name of Agency**	
2012	**Lovely Housing**	38.217391
	Other	32.154545
	Your Estate	35.691304
2013	**Lovely Housing**	38.526923
	Other	37.375000
	Your Estate	39.077564

1.4 Task 4 —— 5 points

> 题目：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

> 题目：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

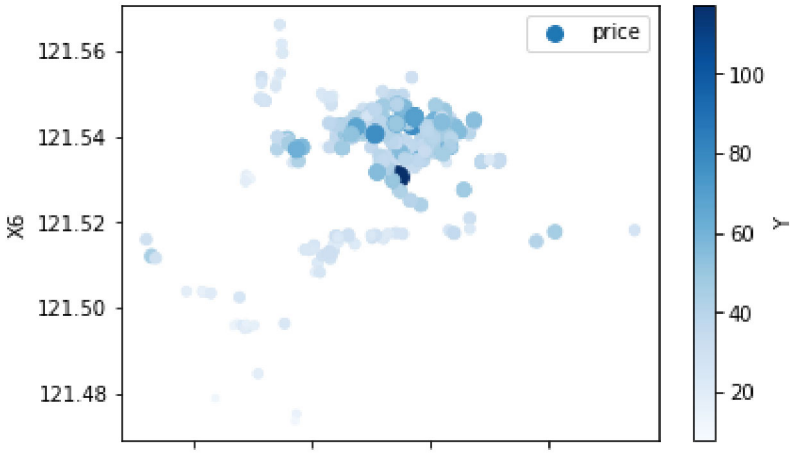
> Select the features and labels (X and y) from your dataset. Make sure that your selection is reasonable. Split the data into two datasets (training and testing).

2.1.1 增加属性

根据$X5$（纬度）和$X6$（经度），房子地理位置分布图如下所示。

In [9]:

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



> 如上图所示，房价明显与房子距市中心的距离有关。因此为了增强特征与房价的相关性，并且降低经纬度之间的多重共线性，我们采用下列公式计算市中心位置。并根据市中心位置增加$X7$列为房子与市中心的距离。
>
> $$center_x = \frac{\sum x_i P}{\sum P}$$
> $$center_y = \frac{\sum y_i P}{\sum P}$$
>
> 其中 (x_i, y_i) 代表经纬度所表示的位置信息, 而 P 是房价.

In [10]:

```
# 计算市中心位置
center_x = sum(df['X5'] * df['Y']) / df['Y'].sum()
center_y = sum(df['X6'] * df['Y']) / df['Y'].sum()
# 一单位经纬度大约为100km。因此为了表述更加直观，将X7列的数据增加100000倍
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 挑选特征

In [11]:

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

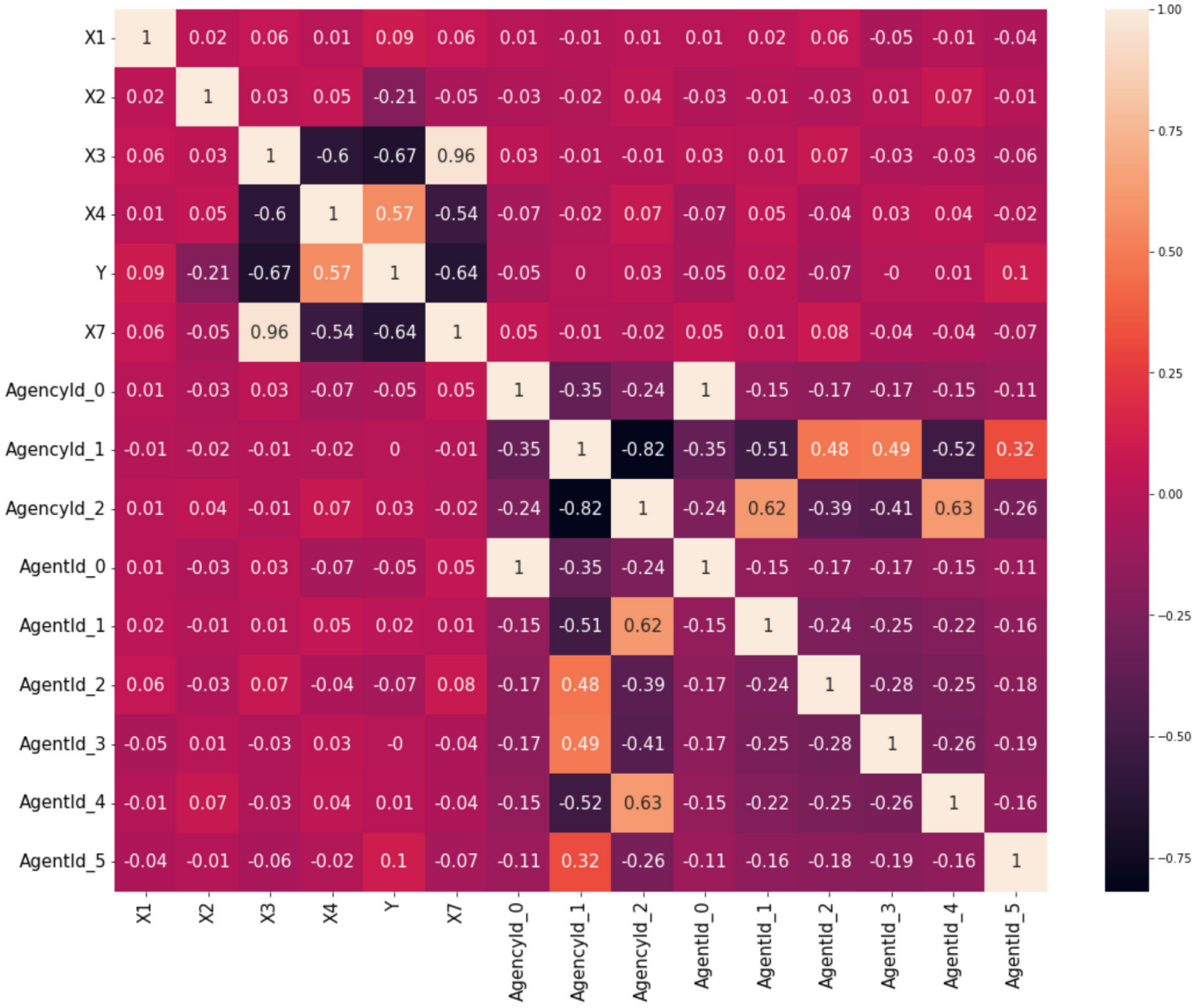
```
df.head()
```

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]:

```
# 计算机相关系数矩阵
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);
```



根据目标Y列与其他特征的相关系数，选择相关系数大于0.5的特征列，再去除掉$X5$（纬度）和$X6$（经度）两列，最终所选特征为$X3$,$X4$和$X7$列。

In [13]:

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

Out[13]:

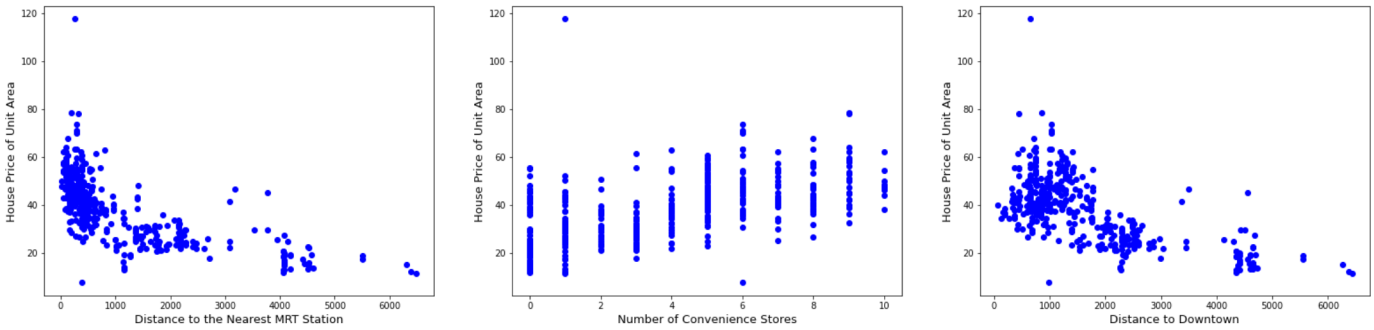
	X3	X4	X7	Y
0	2469.64500	4	2778.053397	23.8
1	289.32480	5	1282.246917	46.2
2	4510.35900	1	4646.003385	13.2
3	1160.63200	0	2262.259333	13.7
4	90.45606	9	744.690602	59.0

2.1.3 去除离群点

In [14]:

```
# 绘制不同特征与房价的散点图
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)
```

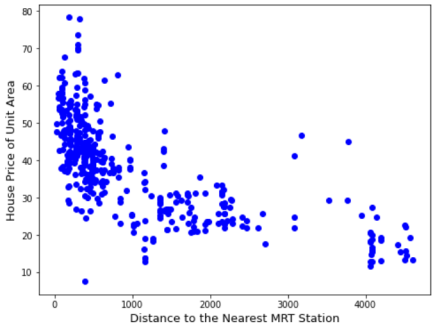


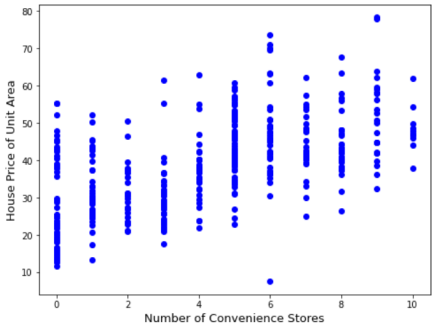
假设数据是正态分布的，因此我们可以根据3σ原则去除离群点

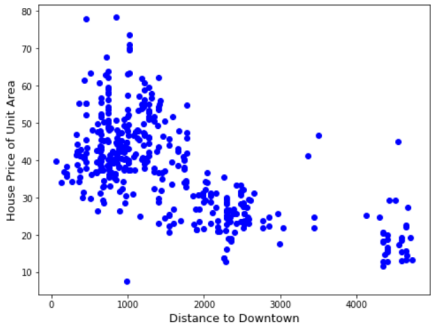
In [15]:

```
# 去除离群点
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)
```







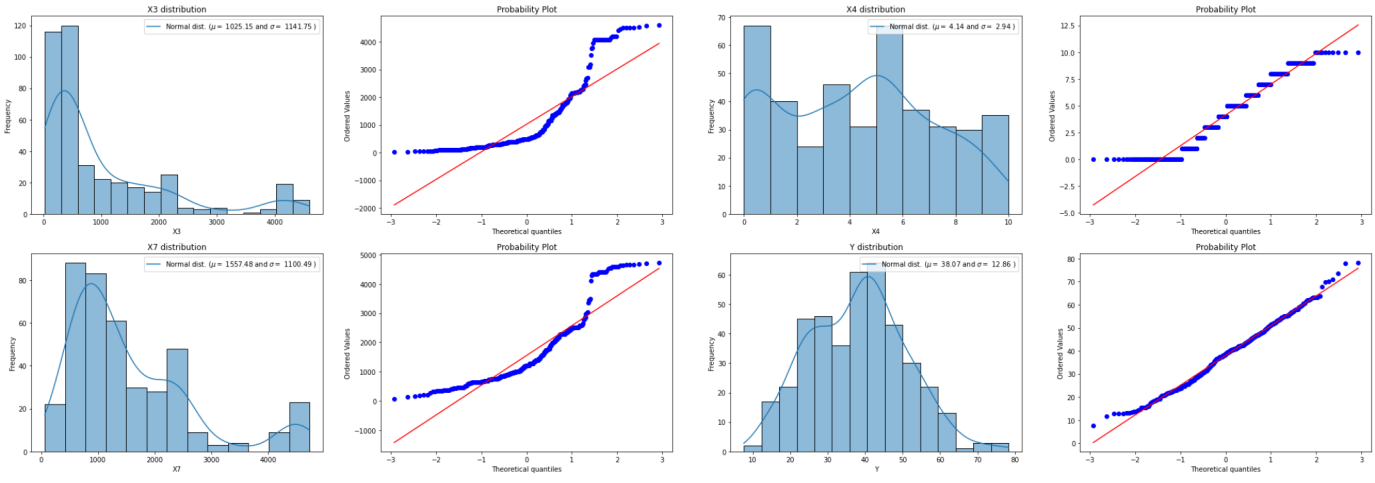
2.1.4 调整变量

In [16]:

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

In [17]:

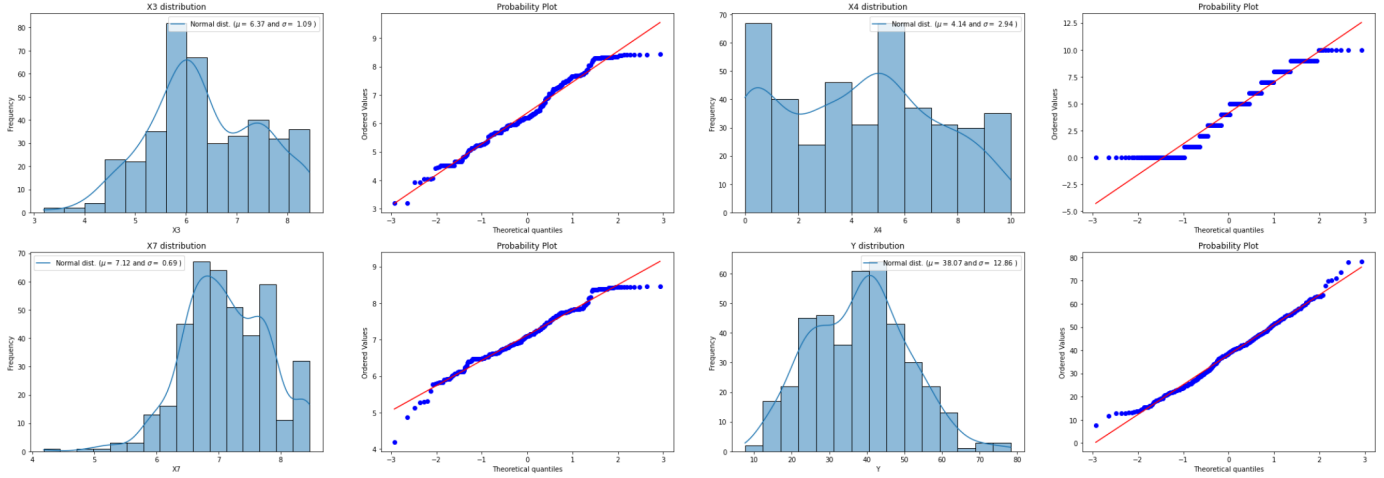
```
def plt_distribution(data):
  """绘制直方图和概率分布图"""
  plt.figure(figsize=(36,12))

  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)
```



根据上图我们可知$X3$和$X7$都较为偏离正态分布，因此我们对它们使用对数处理进行纠正。

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

> 题目：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.

评估模型使用如下指标：

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

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

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

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

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

其中 $\hat{y}_i$ 是房价预测值，y_i 真实值以及 $\overline{y}$ 是平均值。**模型越好，其中MAE、MAPE、SMAPE和MSE的值越小，而R2的值越大。**

In [76]:

```
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):
  """将df数据按照random_state切分为训练集与测试集"""
  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)

  # 标准化
  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):
  '''将结果拟合loop次获得模型的评价指标的平均值'''
  metrics_train = [0,0,0,0,0]
  metrics_test = [0,0,0,0,0]
  for i in range(loop):
    if isAutoML:
      # 分隔数据
      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]:.2f
  print(f"Test accurary:\t{metrics_test[0]:.2f}\t{metrics_test[1]:.2f}%\t{metrics_test[2]:.2f}\t
```

定义一个类便于使用网格搜索超参数。

In [49]:
```
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 线性模型

> 使用线性模型需要经过Task 5中对数据的处理。因此倘若按顺序执行，那么将不会有问题。

In [50]:
```
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 [51]:
```
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: (305, 2)
X test shape: (102, 2)
y train shape: (305,)
y test shape: (102,)
```

In [52]:
```
# 网格搜索最优超参数值
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: 9.1057
Lasso:
Best Params: {'alpha': 0.0009, 'max_iter': 10000, 'random_state': 1},      RMSE: 9.0359
ElasticNet:
Best Params: {'alpha': 0.005, 'l1_ratio': 0.08, 'max_iter': 10000, 'random_state': 1},      RMSE:
9.0357
```

In [53]:
```
%%time
```

```
# 获得线性模型预测结果
# 其中改变模型的random_state20次，将获得评价指标的平均值
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: 6.48    20.28%  0.52    18.69%  79.74
Test accurary:  6.56    21.51%  0.51    19.31%  79.56

Ridge:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.60    20.83%  0.51    18.53%  82.65
Test accurary:  6.41    19.46%  0.51    18.16%  75.14

Lasso:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.48    20.48%  0.52    18.76%  78.60
Test accurary:  6.63    20.92%  0.50    19.08%  84.65

ElasticNet:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.37    19.96%  0.54    18.41%  75.75
Test accurary:  6.90    22.36%  0.44    20.39%  93.30

BayesianRidge:
                MAE     MAPE    R2      SMAPE   MSE
Train accurary: 6.42    20.50%  0.53    18.65%  77.54
Test accurary:  6.73    20.84%  0.47    19.55%  86.46
CPU times: user 1.25 s, sys: 12 ms, total: 1.26 s
Wall time: 1.26 s
```

以上五类线性模型的结果都比较接近，这表明提升模型的复杂度并不能很好提升模型，由此可知模型出现了过拟合，并且产生过拟合的原因是数据不足。

2.2.2 其他模型

> 在这一部分，为了降低数据的复杂性和减少数据多重共线性的影响，我们使用PCA（主成分分析）方法对数据进行降维。

In [54]:

```
# 获取最初数据
df = source.copy()

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

# 去除离群点
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 [55]:

```
# PCA降维
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 [56]:

```
# 导入模型
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from lightgbm import LGBMRegressor
```

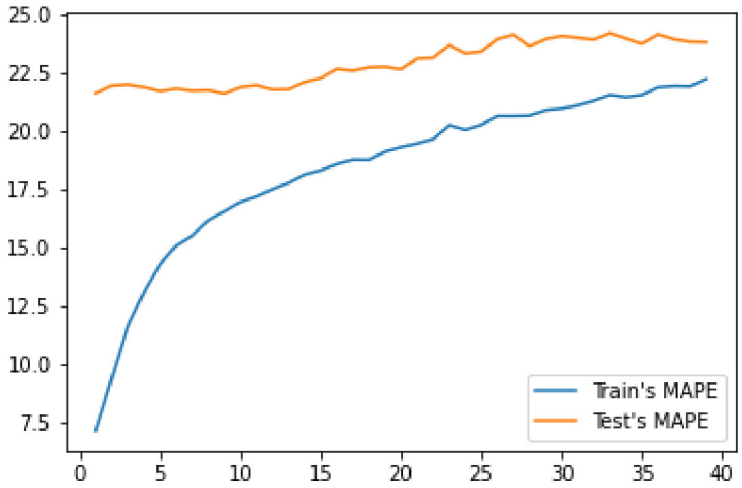
这三个模型相对此次的数据规模而言看过于复杂，因此为了防止过拟合，选用如下观察训练曲线的方式进行调参。

In [57]:

```
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[57]:

```
<matplotlib.legend.Legend at 0x7fa85fa96b10>
```



如上，为了平衡欠拟合与过拟合，这里*min_samples_lea*f的值应该是16。其他参数调参调整方式同理。为了简化表达，下文中的模型参数已经被调整过了。

In [58]:
```
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: 5.00    14.90%  0.71    13.77%  47.62
Test accurary:  5.86    18.08%  0.60    16.30%  63.66
```

In [59]:
```
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: 6.17    15.64%  0.57    16.47%  70.71
Test accurary:  6.74    17.45%  0.49    18.15%  84.72
```

In [60]:
```
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: 4.56    14.20%  0.76    12.88%  39.64
Test accurary:  5.53    17.05%  0.65    15.31%  56.38
```

如上可知，这三类更为复杂的模型相对线性模型并不能提升很多预测性能，并且出现了更加严重的过拟合。

2.2.3 自动机器学习

In [61]:
```
%%capture
!pip install autogluon
```

In [62]:
```
# 导入模块
# 如果这里导入模块失败，重启内核即可
```

```
from autogluon.tabular import TabularDataset, TabularPredictor
```

In [63]:

```
# 获取最初数据
df = source.copy()

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

# 去除离群点
df = remove_outliers(df, columns=['X2','X3','X4','X5','X6','Y'])

# 分隔数据
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降维
(X_train, X_test) = PCA_process(X_train, X_test, 2)

# 重新合并数据
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 [65]:

```
%%time
# 展示一次自动调参过程
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:                    12585.63 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
        0.1s     = Validation runtime
Fitting model: KNeighborsDist ...
        -6.8915  = Validation score   (-root_mean_squared_error)
        0.0s     = Training   runtime
        0.1s     = Validation runtime
Fitting model: LightGBMXT ...
        -6.4161  = Validation score   (-root_mean_squared_error)
        0.31s    = Training   runtime
        0.03s    = Validation runtime
Fitting model: LightGBM ...
        -6.2829  = Validation score   (-root_mean_squared_error)
        0.21s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: RandomForestMSE ...
        -5.8657  = Validation score   (-root_mean_squared_error)
        0.63s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: CatBoost ...
        -6.3514  = Validation score   (-root_mean_squared_error)
        0.49s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: ExtraTreesMSE ...
        -6.1206  = Validation score   (-root_mean_squared_error)
        0.62s    = Training   runtime
        0.1s     = Validation runtime
Fitting model: NeuralNetFastAI ...
No improvement since epoch 8: early stopping
        -6.5688  = Validation score   (-root_mean_squared_error)
        3.37s    = Training   runtime
        0.06s    = Validation runtime
Fitting model: XGBoost ...
        -7.0584  = Validation score   (-root_mean_squared_error)
        0.53s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: NeuralNetTorch ...
        -6.7793  = Validation score   (-root_mean_squared_error)
        2.43s    = Training   runtime
        0.01s    = Validation runtime
Fitting model: LightGBMLarge ...
        -6.5484  = Validation score   (-root_mean_squared_error)
        0.56s    = Training   runtime
        0.0s     = Validation runtime
Fitting model: WeightedEnsemble_L2 ...
        -5.7174  = Validation score   (-root_mean_squared_error)
        1.04s    = Training   runtime
```

```
        0.01s     = Validation runtime
AutoGluon training complete, total runtime = 11.8s ... Best model: "WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor = TabularPredictor.load("best_model/")
CPU times: user 9 s, sys: 389 ms, total: 9.39 s
Wall time: 11.8 s
```

In [77]:

```
%%time
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 14s, sys: 3.51 s, total: 1min 18s
Wall time: 1min 20s
```

由此可见，自动机器学习所得的模型虽然存在，但是预测性能更佳，。**我们将其选择为此题的最终模型。**

2.3 Task 7 – 10 points

> 题目：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。
  - 十分复杂的*WeightedEnsemble_L2*模型的训练集MAPE为9.24%，测试集MAPE为14.38%，直观说明出现了比较严重的过拟合。并且在整个训练过程中，模型从线性模型到随机森林模型到自动机器学习模型，模型越复杂，过拟合越发严重。因此，过拟合的一部分原因是模型过于复杂。
  - 另外一个可解释的存在过拟合的现象是，不论上述哪种模型，只要随机改变训练集和测试集的分割，模型的结果会出现比较大的波动。这也就是说，数据量太少，不论哪种模型都不能完成充分的训练，从而导致了过拟合。所以，过拟合另一个原因是数据样本太少。
- Is your model underfitting?
  - Cannot say.
  - 我们可以说模型不是过拟合的。因为从模型与数据的匹配度而言，*WeightedEnsemble_L2*模型必然能够满足数据的要求。并且模型从线性模型到随机森林模型到自动机器学习模型，模型在测试集上的MAPE逐渐降低，但是过拟合的程度逐渐加剧。因此增加模型的预测性能将以过拟合增大为代价，所以模型已经达到这个数据规模所能训练出的模型的上限。因此可以说模型不存在欠拟合。
  - 我们也可以说模型是存在欠拟合的。因为从PCA降维的过程便可以看出，数据的特征几乎可以完全降维到一列上，所以说数据的特征太少了，并且特征之间的多重共线性比较严重。特征维度过少，导致拟合的函数无法满足训练集，误差较大。因此模型存在欠拟合。