```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

选择文件 SeoulBikeData.csv

- **SeoulBikeData.csv**(text/csv) - 604169 bytes, last modified: 2021/12/6 - 100% done

```
Saving SeoulBikeData.csv to SeoulBikeData.csv
User uploaded file "SeoulBikeData.csv" with length 604169 bytes
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from xgboost.sklearn import XGBRegressor
```

```
data= pd.read_csv('SeoulBikeData.csv', encoding = 'latin-1')
data.head()
```

	ï»¿Date	Rented Bike Count	Hour	Temperature(Â°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0

```
data.describe()
```

	Rented Bike Count	Hour	Temperature(Â°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)
count	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000	8760.000000
mean	704.602055	11.500000	12.882922	58.226256	1.724909	1436.825799	4.073813	0.569111	0.148687
std	644.997468	6.922582	11.944825	20.362413	1.036300	608.298712	13.060369	0.868746	1.128193
min	0.000000	0.000000	-17.800000	0.000000	0.000000	27.000000	-30.600000	0.000000	0.000000
50%	504.500000	11.500000	13.700000	57.000000	1.500000	1698.000000	5.100000	0.010000	0.000000

data preprocessing

```
new_sec = pd.get_dummies(data, columns=['Seasons'],prefix=['season'],drop_first=True)
data = new_sec
data.head()
```

	ï»¿Date	Rented Bike Count	Hour	Temperature(Â°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0

```
new_hol = pd.get_dummies(data, columns=['Holiday'],prefix=['holiday'],drop_first=True)
data = new_hol
data.head()
```

	ï»¿Date	Rented Bike Count	Hour	Temperature(Â°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0

```
new_fuc = pd.get_dummies(data, columns=['Functioning Day'],prefix=['functioning day'],drop_first=True)
data = new_fuc
data.head()
```

	ï»¿Date	Rented Bike Count	Hour	Temperature(Â°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)
0	01/12/2017	254	0	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0
1	01/12/2017	204	1	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0
2	01/12/2017	173	2	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0
3	01/12/2017	107	3	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0
4	01/12/2017	78	4	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0

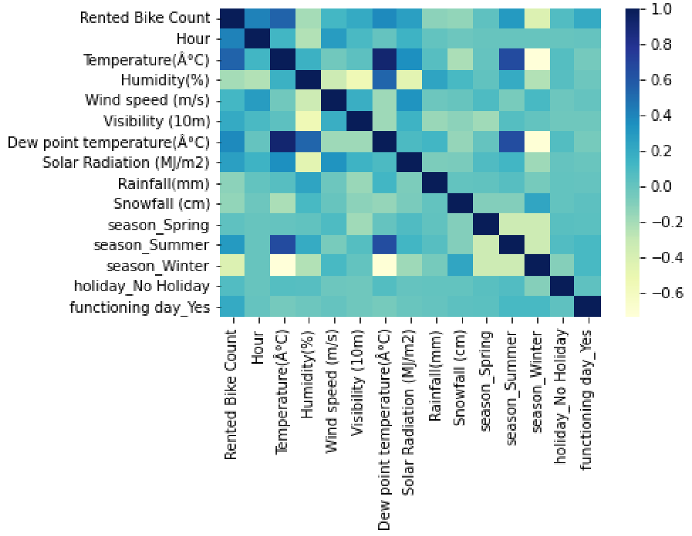
```
data.columns
```

```
Index(['ï»¿Date', 'Rented Bike Count', 'Hour', 'Temperature(Â°C)',
       'Humidity(%)', 'Wind speed (m/s)', 'Visibility (10m)',
       'Dew point temperature(Â°C)', 'Solar Radiation (MJ/m2)', 'Rainfall(mm)',
       'Snowfall (cm)', 'season_Spring', 'season_Summer', 'season_Winter',
       'holiday_No Holiday', 'functioning day_Yes'],
      dtype='object')
```

```
corr = data[['Rented Bike Count', 'Hour', 'Temperature(Â°C)',
       'Humidity(%)', 'Wind speed (m/s)', 'Visibility (10m)',
       'Dew point temperature(Â°C)', 'Solar Radiation (MJ/m2)', 'Rainfall(mm)',
       'Snowfall (cm)', 'season_Spring', 'season_Summer', 'season_Winter',
       'holiday_No Holiday', 'functioning day_Yes']].corr()
corr
```

```
sns.heatmap(corr, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6192b9ffd0>
```

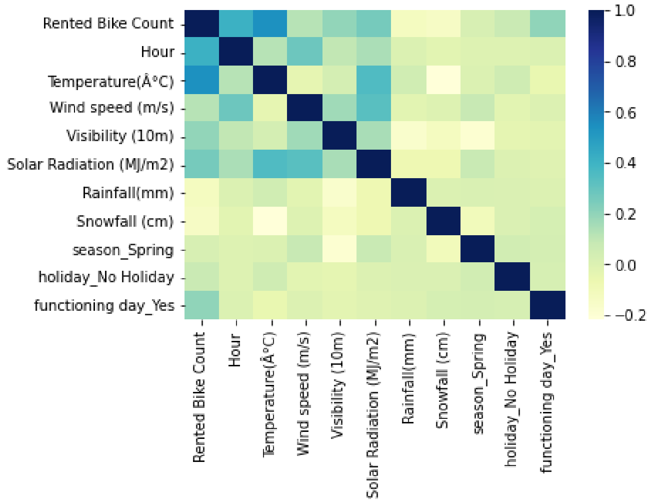


delate Dew point tem, season_summer, season_winter, Humidity

```
data_lr = data.drop(['Dew point temperature(Â°C)','season_Summer', 'season_Winter','Humidity(%)'], axis = 1)
corr = data_lr[['Rented Bike Count', 'Hour', 'Temperature(Â°C)',
       'Wind speed (m/s)', 'Visibility (10m)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)',
       'season_Spring','holiday_No Holiday', 'functioning day_Yes']].corr()
corr
```

```
sns.heatmap(corr, cmap="YlGnBu")
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f618aa15590>




```
new_hour = pd.get_dummies(data, columns=['Hour'],prefix=['Hour'],drop_first=True)
data = new_hour
data.head()
```

	ï»¿Date	Rented Bike Count	Temperature(Â°C)	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)	seas
0	01/12/2017	254	-5.2	37	2.2	2000	-17.6	0.0	0.0	0.0	
1	01/12/2017	204	-5.5	38	0.8	2000	-17.6	0.0	0.0	0.0	
2	01/12/2017	173	-6.0	39	1.0	2000	-17.7	0.0	0.0	0.0	
3	01/12/2017	107	-6.2	40	0.9	2000	-17.6	0.0	0.0	0.0	
4	01/12/2017	78	-6.0	36	2.3	2000	-18.6	0.0	0.0	0.0	

```
new_tem = pd.cut(data['Temperature(Â°C)'],4,labels=['cold', 'cool', 'warm', 'hot'])
data['Temperature(Â°C)'] = new_tem
new_temp = pd.get_dummies(data, columns=['Temperature(Â°C)'],prefix=['Tem'],drop_first=True)
data = new_temp
data.head()
```

	ï»¿Date	Rented Bike Count	Humidity(%)	Wind speed (m/s)	Visibility (10m)	Dew point temperature(Â°C)	Solar Radiation (MJ/m2)	Rainfall(mm)	Snowfall (cm)	season_Spring	season_
0	01/12/2017	254	37	2.2	2000	-17.6	0.0	0.0	0.0	0	
1	01/12/2017	204	38	0.8	2000	-17.6	0.0	0.0	0.0	0	
2	01/12/2017	173	39	1.0	2000	-17.7	0.0	0.0	0.0	0	
3	01/12/2017	107	40	0.9	2000	-17.6	0.0	0.0	0.0	0	
4	01/12/2017	78	36	2.3	2000	-18.6	0.0	0.0	0.0	0	

```
data.columns
```

```
Index(['ï»¿Date', 'Rented Bike Count', 'Humidity(%)', 'Wind speed (m/s)',
       'Visibility (10m)', 'Dew point temperature(Â°C)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)',
       'season_Spring', 'season_Summer', 'season_Winter', 'holiday_No Holiday',
       'functioning day_Yes', 'Hour_1', 'Hour_2', 'Hour_3', 'Hour_4', 'Hour_5',
       'Hour_6', 'Hour_7', 'Hour_8', 'Hour_9', 'Hour_10', 'Hour_11', 'Hour_12',
       'Hour_13', 'Hour_14', 'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18',
       'Hour_19', 'Hour_20', 'Hour_21', 'Hour_22', 'Hour_23', 'Tem_cool',
       'Tem_warm', 'Tem_hot'],
      dtype='object')
```

```
len(data.columns)
```

```
40
```

```
x = data[['Humidity(%)', 'Wind speed (m/s)',
       'Visibility (10m)', 'Dew point temperature(Â°C)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)',
       'season_Spring', 'season_Summer', 'season_Winter', 'holiday_No Holiday',
       'functioning day_Yes', 'Hour_1', 'Hour_2', 'Hour_3', 'Hour_4', 'Hour_5',
       'Hour_6', 'Hour_7', 'Hour_8', 'Hour_9', 'Hour_10', 'Hour_11', 'Hour_12',
       'Hour_13', 'Hour_14', 'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18',
       'Hour_19', 'Hour_20', 'Hour_21', 'Hour_22', 'Hour_23', 'Tem_cool',
       'Tem_warm', 'Tem_hot']]
y = data['Rented Bike Count']
```

linear regression

```
new_hour = pd.get_dummies(data_lr, columns=['Hour'],prefix=['Hour'],drop_first=True)
data_lr = new_hour
new_tem = pd.cut(data_lr['Temperature(Â°C)'],4,labels=['cold', 'cool', 'warm', 'hot'])
data_lr['Temperature(Â°C)'] = new_tem
new_temp = pd.get_dummies(data_lr, columns=['Temperature(Â°C)'],prefix=['Tem'],drop_first=True)
data_lr = new_temp
```

```
data_lr.columns
```

```
Index(['ï»¿Date', 'Rented Bike Count', 'Wind speed (m/s)', 'Visibility (10m)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)',
       'season_Spring', 'holiday_No Holiday', 'functioning day_Yes', 'Hour_1',
```

```
            'Hour_2', 'Hour_3', 'Hour_4', 'Hour_5', 'Hour_6', 'Hour_7', 'Hour_8',
            'Hour_9', 'Hour_10', 'Hour_11', 'Hour_12', 'Hour_13', 'Hour_14',
            'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18', 'Hour_19', 'Hour_20',
            'Hour_21', 'Hour_22', 'Hour_23', 'Tem_cool', 'Tem_warm', 'Tem_hot'],
           dtype='object')
```

```
x_lr = data_lr[['Wind speed (m/s)', 'Visibility (10m)',
       'Solar Radiation (MJ/m2)', 'Rainfall(mm)', 'Snowfall (cm)',
       'season_Spring', 'holiday_No Holiday', 'functioning day_Yes', 'Hour_1',
       'Hour_2', 'Hour_3', 'Hour_4', 'Hour_5', 'Hour_6', 'Hour_7', 'Hour_8',
       'Hour_9', 'Hour_10', 'Hour_11', 'Hour_12', 'Hour_13', 'Hour_14',
       'Hour_15', 'Hour_16', 'Hour_17', 'Hour_18', 'Hour_19', 'Hour_20',
       'Hour_21', 'Hour_22', 'Hour_23', 'Tem_cool', 'Tem_warm', 'Tem_hot']]
```

```
#define cross-validation method to use
cv = KFold(n_splits=10, random_state=1, shuffle=True)
#build multiple linear regression model
lr = LinearRegression()
#use k-fold CV to evaluate model
scores = cross_val_score(lr, x_lr, y, scoring='neg_mean_squared_error',
                         cv=cv, n_jobs=-1)
lr_mse = np.mean(- scores)
print('the mse of linear regression is: ', lr_mse)
```

```
the mse of linear regression is:  157261.75138991658
```

```
#define cross-validation method to use
cv = KFold(n_splits=10, random_state=1, shuffle=True)
#build multiple linear regression model
lr = LinearRegression()
#use k-fold CV to evaluate model
scores = cross_val_score(lr, x_lr, y, scoring='r2',
                         cv=cv, n_jobs=-1)
r2 = np.mean(scores)
print('the r2 of linear regression is: ', r2)
```

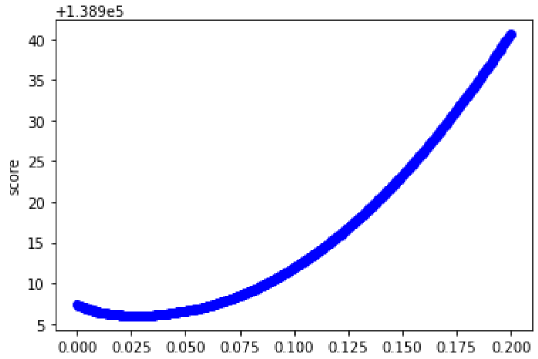
```
the r2 of linear regression is:  0.6215266289297408
```

lasso

```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
#build multiple linear regression model
lasso_mse=[]
for i in np.arange(0,0.2,0.0005):
  lasso = Lasso(alpha = i)
  #use k-fold CV to evaluate model
  scores = cross_val_score(lasso, x, y, scoring='neg_mean_squared_error',
                         cv=cv, n_jobs=-1)
  lasso_mse.append(np.mean(- scores))

plt.plot(list(np.arange(0,0.2,0.0005)), lasso_mse, color='b', linestyle='dashed', marker='o',markerfacecolor='blue', markersize=6)
plt.xlabel('lambda')
plt.ylabel('score')
```

```
Text(0, 0.5, 'score')
```



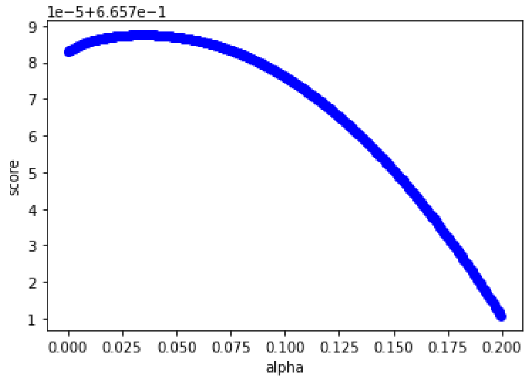

```
print("lambda: ", lasso_mse.index(min(lasso_mse))*0.0005)
print("mse: ", min(lasso_mse))
```

```
lambda:  0.028
mse:  138906.03305971067
```

```
cv = KFold(n_splits=10, random_state=1, shuffle=True)
#build multiple linear regression model
lasso_mse=[]
for i in np.arange(0,0.2,0.0005):
  lasso = Lasso(alpha = i)
  #use k-fold CV to evaluate model
  scores = cross_val_score(lasso, x, y, scoring='r2',
                           cv=cv, n_jobs=-1)
  lasso_mse.append(np.mean(scores))

plt.plot(list(np.arange(0,0.2,0.0005)), lasso_mse, color='b', linestyle='dashed', marker='o',markerfacecolor='blue', markersize=6)
plt.xlabel('lambda')
plt.ylabel('score')
```

```
Text(0, 0.5, 'score')
```




```
max(lasso_mse)
```

```
0.6657875098838415
```

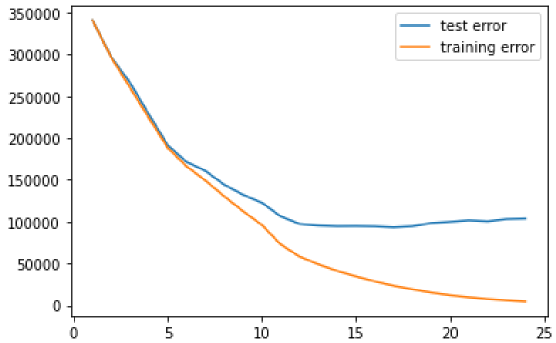
regression tree

```
cver = KFold(n_splits=10, shuffle=True, random_state=0) # K-fold CV environment object
folds = tuple(cver.split(x, y)) # splits into training and testing samples

def GridSearch(estimator, estimator_params):
    return GridSearchCV(estimator, estimator_params, cv=folds,
                        scoring='neg_mean_squared_error', return_train_score=True)

tree = DecisionTreeRegressor(random_state=0)
tree_params = {'max_depth': range(1, 25)}  # dictionary of hyperparameters of the tree
tree_cv = GridSearch(tree, tree_params)
tree_cv.fit(x, y)
tree_cv_results = pd.DataFrame(tree_cv.cv_results_)  # estimation results transformed to a pd.DataFrame

plt.plot(tree_cv_results['param_max_depth'], -tree_cv_results['mean_test_score'], label='test error')
plt.plot(tree_cv_results['param_max_depth'], -tree_cv_results['mean_train_score'], label='training error')
plt.legend()
plt.show()
```




```
tree_cv_results
```

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_max_depth	params	split0_test_score	split1_test_sc
0	0.010374	0.001224	0.002490	0.000128	1	{'max_depth': 1}	−352204.422819	−343757.44
1	0.014235	0.002029	0.002698	0.000103	2	{'max_depth': 2}	−300984.413112	−310015.12
2	0.017231	0.001112	0.003072	0.000870	3	{'max_depth': 3}	−267719.559974	−294516.73
3	0.019841	0.000818	0.002813	0.000349	4	{'max_depth': 4}	−228854.160668	−246528.08
4	0.023203	0.001187	0.002812	0.000265	5	{'max_depth': 5}	−176729.348148	−204569.97
5	0.026405	0.001230	0.002898	0.000376	6	{'max_depth': 6}	−156159.908426	−175102.05
6	0.029575	0.000858	0.002881	0.000161	7	{'max_depth': 7}	−143027.331511	−161269.64
7	0.031988	0.000320	0.002797	0.000098	8	{'max_depth': 8}	−129472.682360	−143001.78
8	0.035998	0.001628	0.002934	0.000384	9	{'max_depth': 9}	−123507.812133	−128775.60
9	0.038365	0.001458	0.002850	0.000173	10	{'max_depth': 10}	−115345.930774	−120793.07
10	0.041356	0.001249	0.002889	0.000094	11	{'max_depth': 11}	−97680.916792	−103960.96
11	0.043690	0.000634	0.002797	0.000028	12	{'max_depth': 12}	−89630.907228	−99339.20
12	0.046876	0.001271	0.002836	0.000076	13	{'max_depth': 13}	−90304.979433	−102873.67
13	0.049285	0.000475	0.002810	0.000032	14	{'max_depth': 14}	−88866.246209	−98348.09
14	0.051950	0.000499	0.002980	0.000373	15	{'max_depth': 15}	−89683.779964	−94431.64
15	0.055348	0.001839	0.002868	0.000022	16	{'max_depth': 16}	−83450.525284	−96781.38

```
print(tree_cv.best_params_, tree_cv.best_score_)
```

```
{'max_depth': 17} -93253.09044873233
```

max_depth: 17 ;mse: 93253.090449

random forest

```
max_depth_values = range(24,40)
n_estimators_values = range(50,51)
rf = RandomForestRegressor(max_features='sqrt', random_state=0)
rf_params = {'n_estimators': n_estimators_values, 'max_depth': max_depth_values}
```

```
rf_cv = GridSearch(rf, rf_params)
rf_cv.fit(x, y)
'''
rf_cv_results = pd.DataFrame(rf_cv.cv_results_)

for depth in max_depth_values:
    results = rf_cv_results[rf_cv_results['param_max_depth'] == depth]
    plt.plot(results['param_n_estimators'], -results['mean_test_score'],
            label='max_depth=%s' % depth)
plt.legend()
plt.show()
'''
```

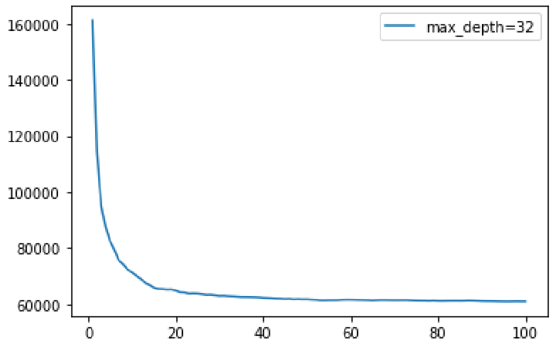
```
'\nrf_cv_results = pd.DataFrame(rf_cv.cv_results_)\n\nfor depth in max_depth_values:\n    results = rf_cv_results[rf_cv_result
s['param_max_depth'] == depth]\n    plt.plot(results['param_n_estimators'], -results['mean_test_score'],\n            label='m
ax_depth=%s' % depth)\nplt.legend()\nplt.show()\n'
```

```
print(rf_cv.best_params_, rf_cv.best_score_)
```

```
{'max_depth': 32, 'n_estimators': 50} -61751.064016737466
```

```
max_depth_values = range(32,33)
n_estimators_values = range(1,101)
rf = RandomForestRegressor(max_features='sqrt', random_state=0)
rf_params = {'n_estimators': n_estimators_values, 'max_depth': max_depth_values}
rf_cv = GridSearch(rf, rf_params)
rf_cv.fit(x, y)
rf_cv_results = pd.DataFrame(rf_cv.cv_results_)

for depth in max_depth_values:
    results = rf_cv_results[rf_cv_results['param_max_depth'] == depth]
    plt.plot(results['param_n_estimators'], -results['mean_test_score'],
            label='max_depth=%s' % depth)
plt.legend()
plt.show()
```




```
print(rf_cv.best_params_, rf_cv.best_score_)
```

```
{'max_depth': 32, 'n_estimators': 96} -60952.22667642568
```

max_depth: 32, mse: 60952.22667642568

xgboost

```
xgbe = XGBRegressor(random_state=0, verbosity = 0)
n_estimators_values = range(50, 51)
xgbe_params = {'n_estimators': n_estimators_values, 'max_depth': range(15,30)}  # dictionary of hyperparameters of the tree
xgbe_cv = GridSearch(xgbe, xgbe_params)
xgbe_cv.fit(x, y)
```

```
'\nxgbe_cv_results = pd.DataFrame(xgbe_cv.cv_results_)  # estimation results transformed to a pd.DataFrame\nfor depth in [15,4
0]:\n    results = xgbe_cv_results[xgbe_cv_results['param_max_depth'] == depth]\n    plt.plot(results['param_n_estimators'], -
results['mean_test_score'],\n            label='max_depth=%s' % depth)\nplt.legend()\nplt.show()\n'
```

```
print(xgbe_cv.best_params_, xgbe_cv.best_score_)
```

```
{'max_depth': 16, 'n_estimators': 50} -61148.434162836
```

```
xgbe = XGBRegressor(random_state=0, verbosity = 0)
n_estimators_values = range(50, 51)
xgbe_params_1 = {'n_estimators': n_estimators_values, 'max_depth': range(1, 15)}  # dictionary of hyperparameters of the tree
xgbe_cv_1 = GridSearch(xgbe, xgbe_params_1)
xgbe_cv_1.fit(x, y)
```

```
GridSearchCV(cv=((array([   0,    1,    2, ..., 8757, 8758, 8759]),
                  array([   9,   15,   20,   22,   33,   38,   39,   42,   44,   48,   49,
         50,   98,  119,  121,  122,  133,  134,  140,  144,  147,  152,
        154,  156,  158,  159,  162,  167,  185,  187,  188,  196,  221,
        229,  233,  234,  244,  247,  253,  257,  273,  274,  277,  296,
        304,  317,  318,  321,  343,  379,  380,  384,  388,  393,  394,
        396,  406,  420,  422,  451,  454,  462,  487,  495,  499,  521,
        526,  528,  532,  541,  543,  565,  574,  578,  597,  633,  6...
       8389, 8393, 8401, 8408, 8417, 8418, 8428, 8430, 8435, 8448, 8467,
       8472, 8473, 8498, 8499, 8523, 8527, 8535, 8539, 8541, 8547, 8559,
       8571, 8574, 8575, 8579, 8607, 8615, 8622, 8625, 8629, 8636, 8652,
       8674, 8683, 8684, 8691, 8707, 8735, 8748]))),
             estimator=XGBRegressor(verbosity=0),
             param_grid={'max_depth': range(1, 15),
                         'n_estimators': range(50, 51)},
             return_train_score=True, scoring='neg_mean_squared_error')
```

```
print(xgbe_cv_1.best_params_, xgbe_cv_1.best_score_)
```

```
{'max_depth': 10, 'n_estimators': 50} -59598.60077625162
```

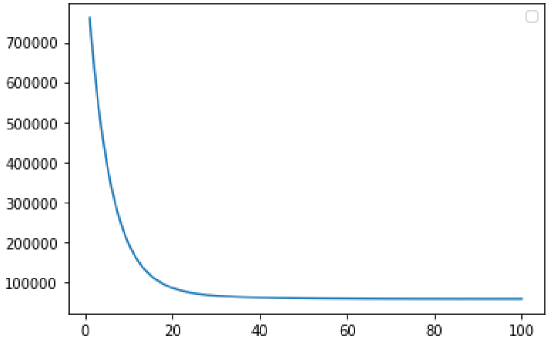
```
xgbe = XGBRegressor(random_state=0, verbosity = 0)
n_estimators_values = range(1, 101)
xgbe_params = {'n_estimators': n_estimators_values, 'max_depth': [10]}  # dictionary of hyperparameters of the tree
xgbe_cv = GridSearch(xgbe, xgbe_params)
xgbe_cv.fit(x, y)

xgbe_cv_results = pd.DataFrame(xgbe_cv.cv_results_)  # estimation results transformed to a pd.DataFrame
```

```
results = xgbe_cv_results['mean_test_score']
plt.plot(n_estimators_values, -results,)
```

```
plt.legend()
plt.show()
```

No handles with labels found to put in legend.

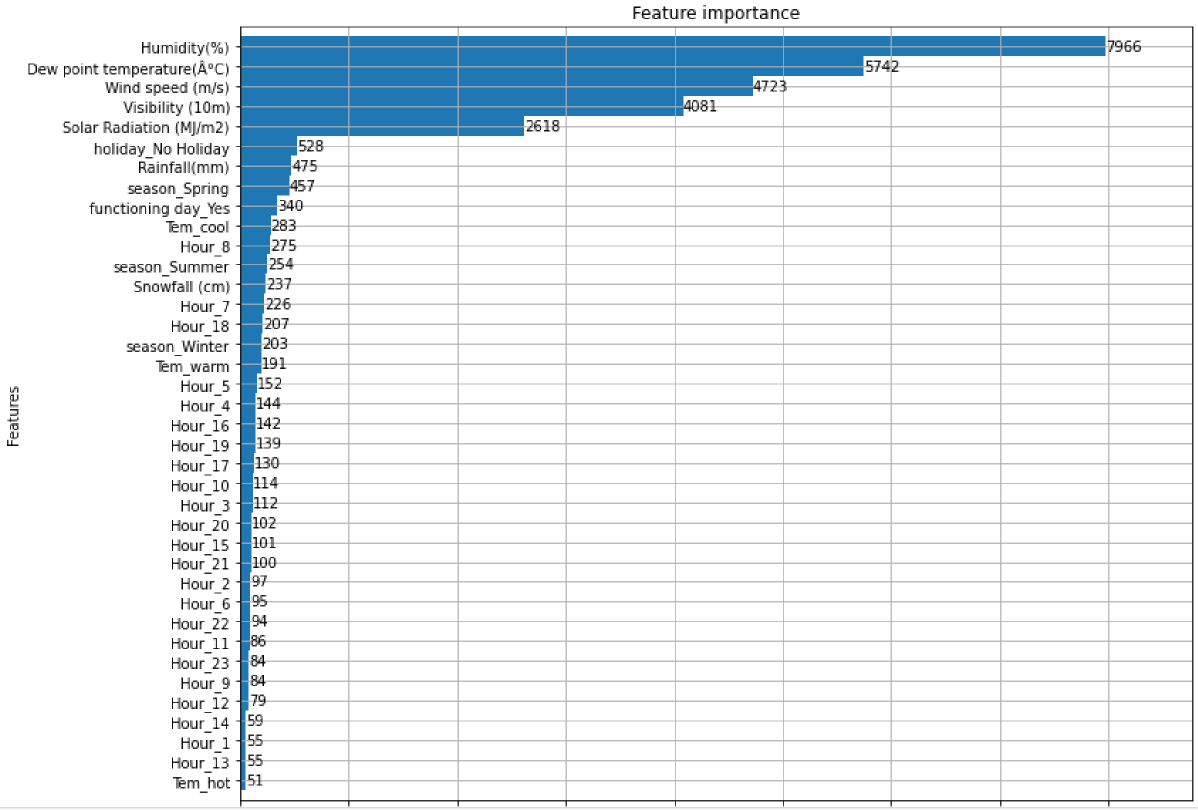



```
print(xgbe_cv.best_params_, xgbe_cv.best_score_)
```

{'max_depth': 10, 'n_estimators': 92} -57628.67093667293

```
from xgboost import plot_importance
xgbe_best = XGBRegressor(random_state=0, verbosity = 0, n_estimators = 92, max_depth = 10)
xgbe_best.fit(x,y)
```

```
ax = plot_importance(xgbe_best, height = 1)
fig = ax.figure
fig.set_size_inches(12, 10)
plt.show()
```



Feature importance
Features
Humidity(%) 7966
Dew point temperature(Â°C) 5742
Wind speed (m/s) 4723
Visibility (10m) 4081
Solar Radiation (MJ/m2) 2618
holiday_No Holiday 528
Rainfall(mm) 475
season_Spring 457
functioning day_Yes 340
Tem_cool 283
Hour_8 275
season_Summer 254
Snowfall (cm) 237
Hour_7 226
Hour_18 207
season_Winter 203
Tem_warm 191
Hour_5 152
Hour_4 144
Hour_16 142
Hour_19 139
Hour_17 130
Hour_10 114
Hour_3 112
Hour_20 102
Hour_15 101
Hour_21 100
Hour_2 97
Hour_6 95
Hour_22 94
Hour_11 86
Hour_23 84
Hour_9 84
Hour_12 79
Hour_14 59
Hour_1 55
Hour_13 55
Tem_hot 51

0 秒 完成时间：15:59