In [3]:
```python
#importing data libraries
import numpy as np
import pandas as pd

import matplotlib
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
%matplotlib inline

import seaborn as sns
sns.set(style="white",color_codes=True)
sns.set(font_scale=1.5)

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import r2_score, mean_squared_error
from sklearn import metrics
from math import sqrt

%matplotlib inline
```

In [4]:
```python
#Importing dataset
#Segregating Data
#Predictors: 'relative_compactness', 'surface_area', 'wall_area', 'roof_area', 'over
#Response Variables: Heating Load and Cooling Load

df=pd.read_csv("C:\\Users\\Pooja Agarwal\\Downloads\\ENB2012_data.csv")
df.columns = ['relative_compactness', 'surface_area', 'wall_area', 'roof_area', 'ove
               'orientation', 'glazing_area', 'glazing_area_distribution', 'heating
df=df.reset_index()
df
```

Out[4]:

	index	relative_compactness	surface_area	wall_area	roof_area	overall_height	orientation	gla
0	0	0.98	514.5	294.0	110.25	7.0	2	
1	1	0.98	514.5	294.0	110.25	7.0	3	
2	2	0.98	514.5	294.0	110.25	7.0	4	
3	3	0.98	514.5	294.0	110.25	7.0	5	
4	4	0.90	563.5	318.5	122.50	7.0	2	
...	...	...	...	...	...	...	...	
763	763	0.64	784.0	343.0	220.50	3.5	5	
764	764	0.62	808.5	367.5	220.50	3.5	2	
765	765	0.62	808.5	367.5	220.50	3.5	3	
766	766	0.62	808.5	367.5	220.50	3.5	4	
767	767	0.62	808.5	367.5	220.50	3.5	5	

768 rows × 11 columns

In [5]:
```
df.describe()
```

Out[5]:

	index	relative_compactness	surface_area	wall_area	roof_area	overall_height	orient
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.00000	768.0(
mean	383.500000	0.764167	671.708333	318.500000	176.604167	5.25000	3.5(
std	221.846794	0.105777	88.086116	43.626481	45.165950	1.75114	1.1'
min	0.000000	0.620000	514.500000	245.000000	110.250000	3.50000	2.0(
25%	191.750000	0.682500	606.375000	294.000000	140.875000	3.50000	2.7!
50%	383.500000	0.750000	673.750000	318.500000	183.750000	5.25000	3.5(
75%	575.250000	0.830000	741.125000	343.000000	220.500000	7.00000	4.2!
max	767.000000	0.980000	808.500000	416.500000	220.500000	7.00000	5.0(

In [6]:
```
df.shape
```

Out[6]:
```
(768, 11)
```

In [7]:
```
#Check for null
df.isnull().sum()
```

Out[7]:
```
index                        0
relative_compactness         0
surface_area                 0
wall_area                    0
roof_area                    0
overall_height               0
orientation                  0
glazing_area                 0
glazing_area_distribution    0
heating_load                 0
cooling_load                 0
dtype: int64
```

In [8]:
```
#Remove the unneccessary data
df.head()
```

Out[8]:

	index	relative_compactness	surface_area	wall_area	roof_area	overall_height	orientation	glazi
0	0	0.98	514.5	294.0	110.25	7.0	2	
1	1	0.98	514.5	294.0	110.25	7.0	3	
2	2	0.98	514.5	294.0	110.25	7.0	4	
3	3	0.98	514.5	294.0	110.25	7.0	5	
4	4	0.90	563.5	318.5	122.50	7.0	2	

In [9]:
```
#Check Null again
df.isnull().sum()
```

Out[9]:
```
index                        0
relative_compactness         0
surface_area                 0
```

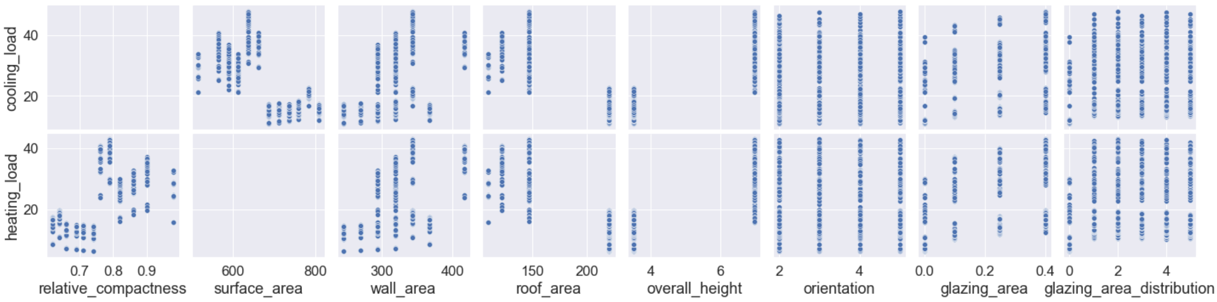
```
wall_area                    0
roof_area                    0
overall_height               0
orientation                  0
glazing_area                 0
glazing_area_distribution    0
heating_load                 0
cooling_load                 0
dtype: int64
```

In [43]:
```
#Check the datatypes in the csv
df.dtypes
```

Out[43]:
```
index                          int64
relative_compactness         float64
surface_area                 float64
wall_area                    float64
roof_area                    float64
overall_height               float64
orientation                    int64
glazing_area                 float64
glazing_area_distribution      int64
heating_load                 float64
cooling_load                 float64
dtype: object
```

In [44]:
```
# Correlation between inputs and outputs
plt.figure(figsize=(5,5))
sns.pairplot(data=df, y_vars=['cooling_load','heating_load'],x_vars=['relative_compa
                     'orientation', 'glazing_area', 'glazing_area_distribution',])
plt.show()
```

```
<Figure size 360x360 with 0 Axes>
```



In [47]:
```
#nr = Normalizer(copy=False)

X = df.drop(['heating_load','cooling_load'], axis=1)
#X = nr.fit_transform(X)
y = df[['heating_load','cooling_load']]
```

In [48]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_st
```

In [49]:
```
#Import decision tree regressor
from sklearn.tree import DecisionTreeRegressor
# Create decision tree model
dt_model = DecisionTreeRegressor(random_state=2)
# Apply the model
dt_model.fit(X_train, y_train)
# Predicted value
y_pred1 = dt_model.predict(X_test)
```

In [50]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_pred1)
```

```
0.9588327472130773
```

Out[50]:

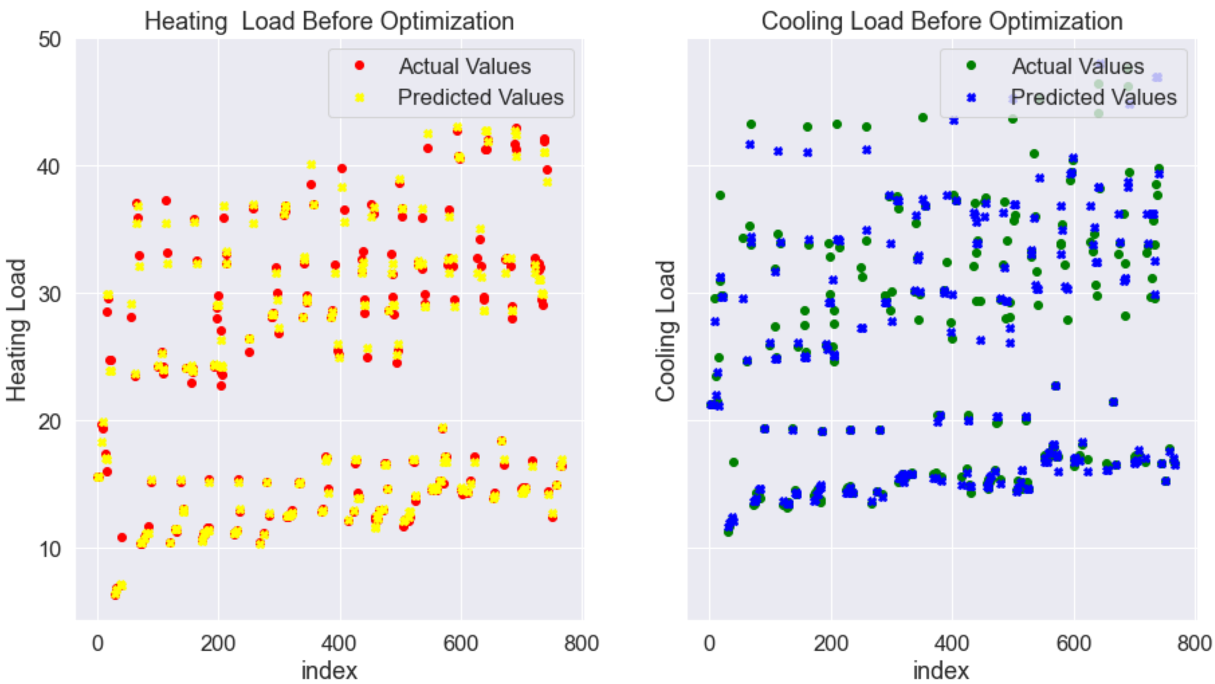
In [52]:

```
f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
#Visualize the heating load output before optimization
ax1.plot(X_test['index'],y_test['heating_load'],'o',color='red',label = 'Actual Valu
ax1.plot(X_test['index'],y_pred1[:,0],'X',color='yellow',label = 'Predicted Values')
ax1.set_xlabel('index')
ax1.set_ylabel('Heating Load')
ax1.set_title('Heating  Load Before Optimization')
ax1.legend(loc = 'upper right')

#Visualize the cooling load output before optimization
ax2.plot(X_test['index'],y_test['cooling_load'].values,'o',color='green',label = 'Ac
ax2.plot(X_test['index'],y_pred1[:,1],'X',color='blue',label = 'Predicted Values')
ax2.set_xlabel('index')
ax2.set_ylabel('Cooling Load')
ax2.set_title('Cooling Load Before Optimization')
ax2.legend(loc = 'upper right')

ax1.figure.set_size_inches(15, 8)


plt.show()
```



In [53]:

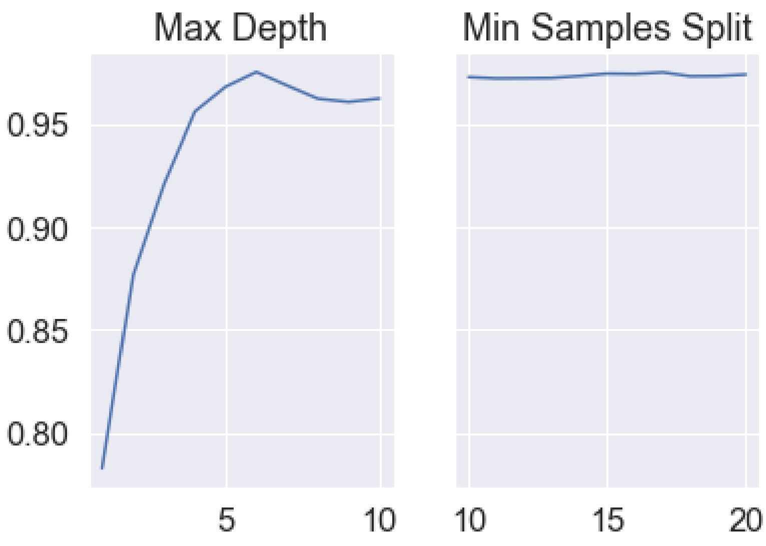
```
# Finding the best decision tree optimization parameters

f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
# Max Depth
dt_acc = []
dt_depth = range(1,11)
for i in dt_depth:
    dt = DecisionTreeRegressor(random_state=2, max_depth=i)
    dt.fit(X_train, y_train)
    dt_acc.append(dt.score(X_test, y_test))
ax1.plot(dt_depth,dt_acc)
ax1.set_title('Max Depth')

#Min Samples Split
dt_acc = []
dt_samples_split = range(10,21)
for i in dt_samples_split:
    dt = DecisionTreeRegressor(random_state=2, min_samples_split=i)
```

```
    dt.fit(X_train, y_train)
    dt_acc.append(dt.score(X_test, y_test))
ax2.plot(dt_samples_split,dt_acc)
ax2.set_title('Min Samples Split')

plt.show()
```

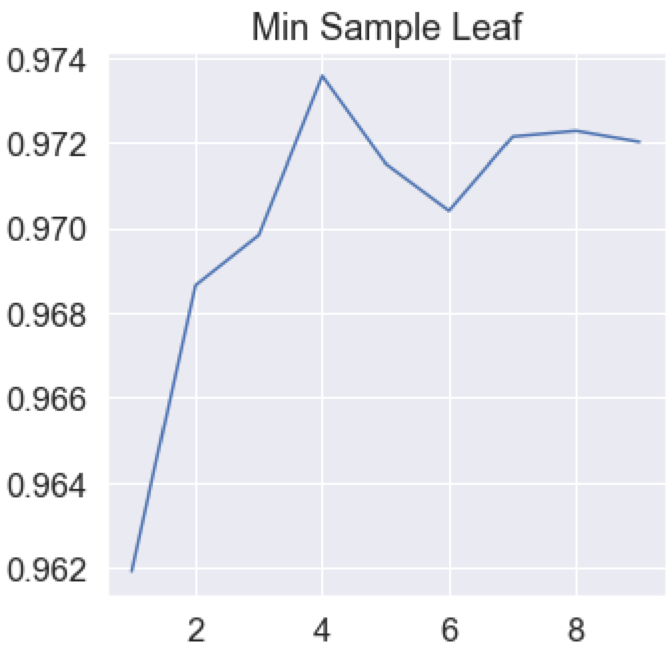


In [54]:

```
#Min Sample Leaf
plt.figure(figsize = (5,5))
dt_acc = []
dt_samples_leaf = range(1,10)
for i in dt_samples_leaf:
    dt = DecisionTreeRegressor(random_state=123, min_samples_leaf=i)
    dt.fit(X_train, y_train)
    dt_acc.append(dt.score(X_test, y_test))

plt.plot(dt_samples_leaf,dt_acc)
plt.title('Min Sample Leaf')

plt.show()
```



In [55]:

```
# Decision tree optimization parameters
from sklearn.model_selection import GridSearchCV
parameters = {'max_depth' : [7,8,9],
              'min_samples_split': [16,17,18],
```

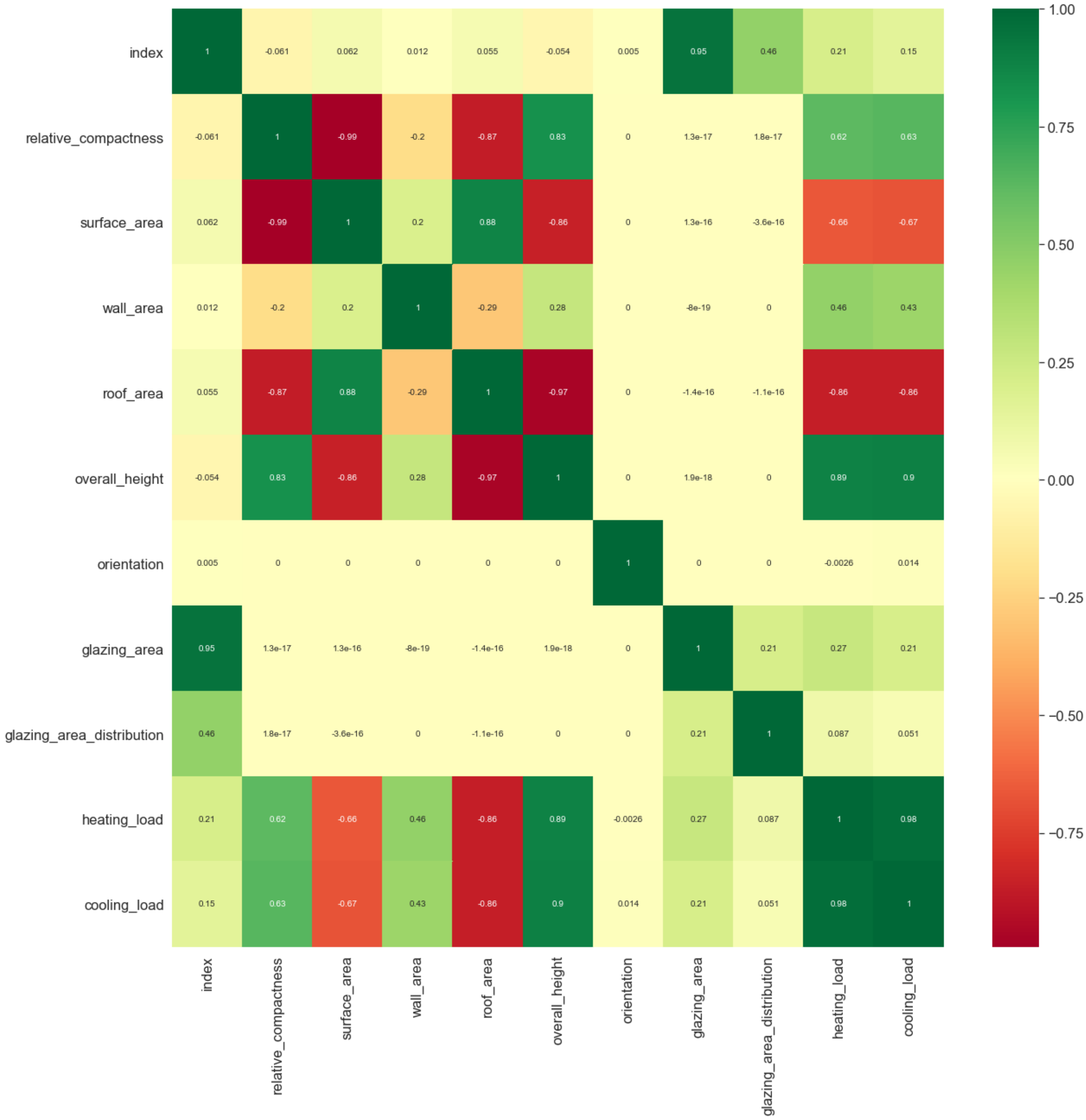
```
                'min_samples_leaf' : [6,7,8]}


#Create new model using the GridSearch
dt_random = GridSearchCV(dt_model, parameters)
```

In [81]:
```
dt_random.fit(X_train, y_train)
```

Out[81]:
```
GridSearchCV(estimator=DecisionTreeRegressor(random_state=2),
             param_grid={'max_depth': [7, 8, 9], 'min_samples_leaf': [6, 7, 8],
                         'min_samples_split': [16, 17, 18]})
```

In [58]:
```
corrmat = df.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(20,20))
g=sns.heatmap(df[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```



In [70]:
```
dt_random.best_params_
```

Out[70]:
```
{'max_depth': 8, 'min_samples_leaf': 6, 'min_samples_split': 16}
```

In [71]:
```
dt_random.best_score_
```

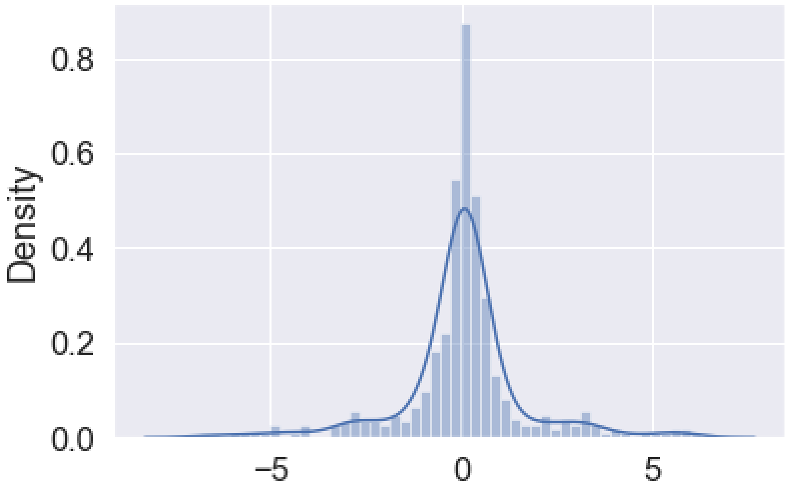
Out[71]:
```
0.9764044281547335
```

In [72]:
```
predictions=dt_random.predict(X_test)
```

In [73]:
```
sns.distplot(y_test-predictions)
```

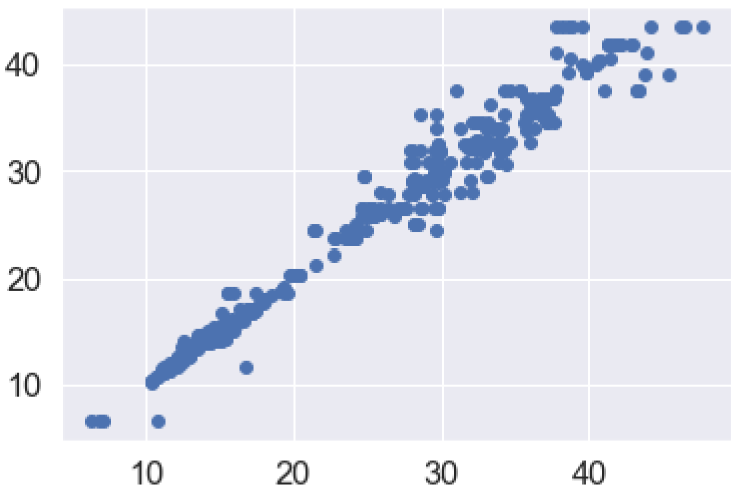
```
C:\Users\Pooja Agarwal\anaconda3\lib\site-packages\seaborn\distributions.py:2551: Fu
tureWarning: `distplot` is a deprecated function and will be removed in a future ver
sion. Please adapt your code to use either `displot` (a figure-level function with s
imilar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[73]: <AxesSubplot:ylabel='Density'>



In [74]:
```
plt.scatter(y_test,predictions)
```

Out[74]: <matplotlib.collections.PathCollection at 0x1f90db90d30>



In [75]:
```
print('MAE:', metrics.mean_absolute_error(y_test, predictions))
print('MSE:', metrics.mean_squared_error(y_test, predictions))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
MAE: 0.9407189110793008
MSE: 2.5984362698870664
RMSE: 1.6119665846062277
```

In [77]:
```
import statsmodels.formula.api as smf
data2=df.copy()
lm1 = smf.ols(formula='heating_load ~ relative_compactness + surface_area + wall_are
```

In [78]:
```
lm1.summary()
```

OLS Regression Results

Out[78]:

Dep. Variable:	heating_load	**R-squared:**	0.916
Model:	OLS	**Adj. R-squared:**	0.915
Method:	Least Squares	**F-statistic:**	1187.
Date:	Sun, 17 Jan 2021	**Prob (F-statistic):**	0.00
Time:	00:36:11	**Log-Likelihood:**	-1912.5
No. Observations:	768	**AIC:**	3841.
Df Residuals:	760	**BIC:**	3878.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	84.0145	19.034	4.414	0.000	46.650	121.379
relative_compactness	-64.7740	10.289	-6.295	0.000	-84.973	-44.575
surface_area	-0.0626	0.013	-4.670	0.000	-0.089	-0.036
wall_area	0.0361	0.004	9.386	0.000	0.029	0.044
roof_area	-0.0494	0.008	-6.569	0.000	-0.064	-0.035
overall_height	4.1699	0.338	12.337	0.000	3.506	4.833
orientation	-0.0233	0.095	-0.246	0.805	-0.209	0.163
glazing_area	19.9327	0.814	24.488	0.000	18.335	21.531
glazing_area_distribution	0.2038	0.070	2.914	0.004	0.067	0.341

Omnibus:	18.648	**Durbin-Watson:**	0.654
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	37.708
Skew:	0.044	**Prob(JB):**	6.48e-09
Kurtosis:	4.082	**Cond. No.**	7.63e+15

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 7.82e-24. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [79]:

```
# create a fitted model with all features excluding "orientation"
lm2 = smf.ols(formula='heating_load ~ relative_compactness + surface_area + wall_are
```

In [80]:

```
lm2.summary()
```

Out[80]:

OLS Regression Results

Dep. Variable:	heating_load	**R-squared:**	0.916
Model:	OLS	**Adj. R-squared:**	0.916
Method:	Least Squares	**F-statistic:**	1387.
Date:	Sun, 17 Jan 2021	**Prob (F-statistic):**	0.00

Time:	00:37:10	**Log-Likelihood:**	-1912.5
No. Observations:	768	**AIC:**	3839.
Df Residuals:	761	**BIC:**	3871.
Df Model:	6		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	83.9329	19.019	4.413	0.000	46.597	121.269
relative_compactness	-64.7740	10.283	-6.299	0.000	-84.961	-44.587
surface_area	-0.0626	0.013	-4.673	0.000	-0.089	-0.036
wall_area	0.0361	0.004	9.392	0.000	0.029	0.044
roof_area	-0.0494	0.008	-6.573	0.000	-0.064	-0.035
overall_height	4.1699	0.338	12.345	0.000	3.507	4.833
glazing_area	19.9327	0.813	24.503	0.000	18.336	21.530
glazing_area_distribution	0.2038	0.070	2.916	0.004	0.067	0.341

Omnibus:	18.654	**Durbin-Watson:**	0.654
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	37.740
Skew:	0.044	**Prob(JB):**	6.38e-09
Kurtosis:	4.082	**Cond. No.**	2.82e+16

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.71e-25. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [10]:
```
import statsmodels.formula.api as smf
data2=df.copy()
lm2 = smf.ols(formula='cooling_load ~ relative_compactness + surface_area + wall_are
```

In [11]:
```
lm2.summary()
```

Out[11]:

OLS Regression Results

Dep. Variable:	cooling_load	**R-squared:**	0.888
Model:	OLS	**Adj. R-squared:**	0.887
Method:	Least Squares	**F-statistic:**	859.1
Date:	Sun, 17 Jan 2021	**Prob (F-statistic):**	0.00
Time:	21:33:27	**Log-Likelihood:**	-1979.3
No. Observations:	768	**AIC:**	3975.
Df Residuals:	760	**BIC:**	4012.
Df Model:	7		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Intercept	97.2457	20.765	4.683	0.000	56.483	138.009
relative_compactness	-70.7877	11.225	-6.306	0.000	-92.824	-48.751
surface_area	-0.0661	0.015	-4.519	0.000	-0.095	-0.037
wall_area	0.0225	0.004	5.365	0.000	0.014	0.031
roof_area	-0.0443	0.008	-5.404	0.000	-0.060	-0.028
overall_height	4.2838	0.369	11.618	0.000	3.560	5.008
orientation	0.1215	0.103	1.176	0.240	-0.081	0.324
glazing_area	14.7171	0.888	16.573	0.000	12.974	16.460
glazing_area_distribution	0.0407	0.076	0.534	0.594	-0.109	0.190

Omnibus:	104.668	**Durbin-Watson:**	1.094
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	230.547
Skew:	0.767	**Prob(JB):**	8.65e-51
Kurtosis:	5.203	**Cond. No.**	7.63e+15

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 7.82e-24. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.

In [2]:
```
#R squared sometimes is not a good indicator of fit. R squared (R2 score) will alway
#Adjusted R squared is recommended than R squared in terms of goodness of fit
```

In [ ]:
```

```