```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# load dataset
df = pd.read_csv("creditcard.csv")
df.head(8)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Clas
0	0.00	-1.36	-0.07	2.54	1.38	-0.34	0.46	0.24	0.10	0.36	...	-0.02	0.28	-0.11	0.07	0.13	-0.19	0.13	-0.02	149.62	
1	0.00	1.19	0.27	0.17	0.45	0.06	-0.08	-0.08	0.09	-0.26	...	-0.23	-0.64	0.10	-0.34	0.17	0.13	-0.01	0.01	2.69	
2	1.00	-1.36	-1.34	1.77	0.38	-0.50	1.80	0.79	0.25	-1.51	...	0.25	0.77	0.91	-0.69	-0.33	-0.14	-0.06	-0.06	378.66	
3	1.00	-0.97	-0.19	1.79	-0.86	-0.01	1.25	0.24	0.38	-1.39	...	-0.11	0.01	-0.19	-1.18	0.65	-0.22	0.06	0.06	123.50	
4	2.00	-1.16	0.88	1.55	0.40	-0.41	0.10	0.59	-0.27	0.82	...	-0.01	0.80	-0.14	0.14	-0.21	0.50	0.22	0.22	69.99	
5	2.00	-0.43	0.96	1.14	-0.17	0.42	-0.03	0.48	0.26	-0.57	...	-0.21	-0.56	-0.03	-0.37	-0.23	0.11	0.25	0.08	3.67	
6	4.00	1.23	0.14	0.05	1.20	0.19	0.27	-0.01	0.08	0.46	...	-0.17	-0.27	-0.15	-0.78	0.75	-0.26	0.03	0.01	4.99	
7	7.00	-0.64	1.42	1.07	-0.49	0.95	0.43	1.12	-3.81	0.62	...	1.94	-1.02	0.06	-0.65	-0.42	-0.05	-1.21	-1.09	40.80	

8 rows × 31 columns

```python
# This shows the number of non fraudulent (0) and fraudulent (1) transactions
print(f"Unique values of target variable \n {df['Class'].unique()}")
print(f"Number of samples under each target value \n {df['Class'].value_counts()}")
```

```
Unique values of target variable
 [0 1]
Number of samples under each target value
 0    284315
1       492
Name: Class, dtype: int64
```

```python
# time is irrelevent to classification so it can be removed
```

```
df = df.drop(['Time'], axis=1)
print(f"list of feature names after removing Time column :- \n{df.columns}")
```

```
list of feature names after removing Time column :-
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Amount', 'Class'],
      dtype='object')
```

```
print(f"Dataset info : \n {df.info()}")
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 30 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   V1      284807 non-null  float64
 1   V2      284807 non-null  float64
 2   V3      284807 non-null  float64
 3   V4      284807 non-null  float64
 4   V5      284807 non-null  float64
 5   V6      284807 non-null  float64
 6   V7      284807 non-null  float64
 7   V8      284807 non-null  float64
 8   V9      284807 non-null  float64
 9   V10     284807 non-null  float64
 10  V11     284807 non-null  float64
 11  V12     284807 non-null  float64
 12  V13     284807 non-null  float64
 13  V14     284807 non-null  float64
 14  V15     284807 non-null  float64
 15  V16     284807 non-null  float64
 16  V17     284807 non-null  float64
 17  V18     284807 non-null  float64
 18  V19     284807 non-null  float64
 19  V20     284807 non-null  float64
 20  V21     284807 non-null  float64
 21  V22     284807 non-null  float64
 22  V23     284807 non-null  float64
 23  V24     284807 non-null  float64
 24  V25     284807 non-null  float64
 25  V26     284807 non-null  float64
 26  V27     284807 non-null  float64
 27  V28     284807 non-null  float64
 28  Amount  284807 non-null  float64
 29  Class   284807 non-null  int64  
```

```
dtypes: float64(29), int64(1)
memory usage: 65.2 MB
Dataset info :
 None
```

```
from sklearn.preprocessing import StandardScaler
# amount needs a smaller range of numbers so it needs to be scaled.
df['norm_amount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
fraud_df = df.drop(['Amount'], axis=1)
print(f"test values from Amount column after applying StandardScaler: \n {fraud_df['norm_amount'][0:4]}")
```

```
test values from Amount column after applying StandardScaler:
 0    0.24
1   -0.34
2    1.16
3    0.14
Name: norm_amount, dtype: float64
```

```
# creating variables for train and test.
X = df.drop(['Class'], axis=1)
y = df[['Class']]
```

```
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_state=42)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(199364, 30)
(85443, 30)
(199364, 1)
(85443, 1)
```

The first implemented classifier is a neural network similar to assignment five

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
y_train = np.asarray(y_train).astype('float32').reshape((-1,1))
y_test = np.asarray(y_test).astype('float32').reshape((-1,1))


from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
n_features = X_train.shape[1]

model  = Sequential()
model.add(Dense(15,activation = 'relu',kernel_initializer = 'he_normal',input_shape = (n_features,)))
model.add(Dense(10,activation = 'relu',kernel_initializer = 'he_normal'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy',tf.keras.metrics.AUC()])

model.fit(X_train, y_train,batch_size = 1000 ,epochs=150, verbose=2)
```

```
156/156 - 0s - loss: 0.0025 - accuracy: 0.9993 - auc: 0.9905 - 455ms/epoch - 3ms/step
Epoch 123/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9994 - auc: 0.9954 - 401ms/epoch - 3ms/step
Epoch 124/150
156/156 - 0s - loss: 0.0023 - accuracy: 0.9994 - auc: 0.9922 - 379ms/epoch - 2ms/step
Epoch 125/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9993 - auc: 0.9905 - 389ms/epoch - 2ms/step
Epoch 126/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9994 - auc: 0.9906 - 348ms/epoch - 2ms/step
Epoch 127/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9993 - auc: 0.9938 - 373ms/epoch - 2ms/step
Epoch 128/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9939 - 376ms/epoch - 2ms/step
Epoch 129/150
156/156 - 0s - loss: 0.0021 - accuracy: 0.9994 - auc: 0.9956 - 382ms/epoch - 2ms/step
Epoch 130/150
156/156 - 0s - loss: 0.0023 - accuracy: 0.9994 - auc: 0.9938 - 396ms/epoch - 3ms/step
Epoch 131/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9906 - 450ms/epoch - 3ms/step
Epoch 132/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9955 - 477ms/epoch - 3ms/step
Epoch 133/150
156/156 - 0s - loss: 0.0025 - accuracy: 0.9994 - auc: 0.9920 - 420ms/epoch - 3ms/step
```

```
Epoch 134/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9993 - auc: 0.9905 - 443ms/epoch - 3ms/step
Epoch 135/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9993 - auc: 0.9921 - 411ms/epoch - 3ms/step
Epoch 136/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9922 - 416ms/epoch - 3ms/step
Epoch 137/150
156/156 - 0s - loss: 0.0026 - accuracy: 0.9993 - auc: 0.9919 - 437ms/epoch - 3ms/step
Epoch 138/150
156/156 - 0s - loss: 0.0021 - accuracy: 0.9995 - auc: 0.9956 - 405ms/epoch - 3ms/step
Epoch 139/150
156/156 - 0s - loss: 0.0023 - accuracy: 0.9994 - auc: 0.9905 - 404ms/epoch - 3ms/step
Epoch 140/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9906 - 356ms/epoch - 2ms/step
Epoch 141/150
156/156 - 0s - loss: 0.0024 - accuracy: 0.9993 - auc: 0.9904 - 364ms/epoch - 2ms/step
Epoch 142/150
156/156 - 0s - loss: 0.0025 - accuracy: 0.9994 - auc: 0.9970 - 421ms/epoch - 3ms/step
Epoch 143/150
156/156 - 0s - loss: 0.0023 - accuracy: 0.9994 - auc: 0.9939 - 370ms/epoch - 2ms/step
Epoch 144/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9956 - 390ms/epoch - 3ms/step
Epoch 145/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9939 - 402ms/epoch - 3ms/step
Epoch 146/150
156/156 - 0s - loss: 0.0025 - accuracy: 0.9993 - auc: 0.9938 - 397ms/epoch - 3ms/step
Epoch 147/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9957 - 404ms/epoch - 3ms/step
Epoch 148/150
156/156 - 0s - loss: 0.0023 - accuracy: 0.9994 - auc: 0.9956 - 387ms/epoch - 2ms/step
Epoch 149/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9956 - 403ms/epoch - 3ms/step
Epoch 150/150
156/156 - 0s - loss: 0.0022 - accuracy: 0.9994 - auc: 0.9955 - 402ms/epoch - 3ms/step
<keras.callbacks.History at 0x7fab1cb93d90>
```

The second classifier is the random forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

def random_forest_classifier(X_train, y_train, X_test, y_test):
     # initialize object for DecisionTreeClassifier class
```

```
    rf_classifier = RandomForestClassifier(n_estimators=50)
    # train model by using fit method
    print("Model training starts")
    rf_classifier.fit(X_train, y_train.ravel())
    acc_score = rf_classifier.score(X_test, y_test)
    print(f'Accuracy of model on test dataset : {acc_score}')
    # predict result using test dataset
    y_pred = rf_classifier.predict(X_test)
    # confusion matrix
    print(f"Confusion Matrix : \n {confusion_matrix(y_test, y_pred)}")
    # classification report for f1-score
    print(f"Classification Report : \n {classification_report(y_test, y_pred)}")


# calling random_forest_classifier
random_forest_classifier(X_train, y_train, X_test, y_test)
```

```
Model training starts
Accuracy of model on test dataset : 0.9995903701883126
Confusion Matrix :
 [[85298     9]
 [   26   110]]
Classification Report :
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85307
         1.0       0.92      0.81      0.86       136

    accuracy                           1.00     85443
   macro avg       0.96      0.90      0.93     85443
weighted avg       1.00      1.00      1.00     85443
```

The third is the decision tree classifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report

def decision_tree_classification(X_train, y_train, X_test, y_test):
    # initialize object for DecisionTreeClassifier class
    dt_classifier = DecisionTreeClassifier()
    # train model by using fit method
```

```
    print("Model training starts")
    dt_classifier.fit(X_train, y_train.ravel())
    print("Model training completed")
    acc_score = dt_classifier.score(X_test, y_test)
    print(f'Accuracy of model on test dataset : {acc_score}')
    # predict result using test dataset
    y_pred = dt_classifier.predict(X_test)
    # confusion matrix
    print(f"Confusion Matrix : \n {confusion_matrix(y_test, y_pred)}")
    # classification report for f1-score
    print(f"Classification Report : \n {classification_report(y_test, y_pred)}")
```

```
# calling decision_tree_classification method to train and evaluate model
decision_tree_classification(X_train, y_train, X_test, y_test)
```

```
Model training starts
Model training completed
Accuracy of model on test dataset : 0.999133925541004
Confusion Matrix :
 [[85263    44]
 [   30   106]]
Classification Report :
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85307
         1.0       0.71      0.78      0.74       136

    accuracy                           1.00     85443
   macro avg       0.85      0.89      0.87     85443
weighted avg       1.00      1.00      1.00     85443
```

The beginning of the Regression section of the project.

```
import pandas as pd
import pandas_profiling
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('whitegrid')
```

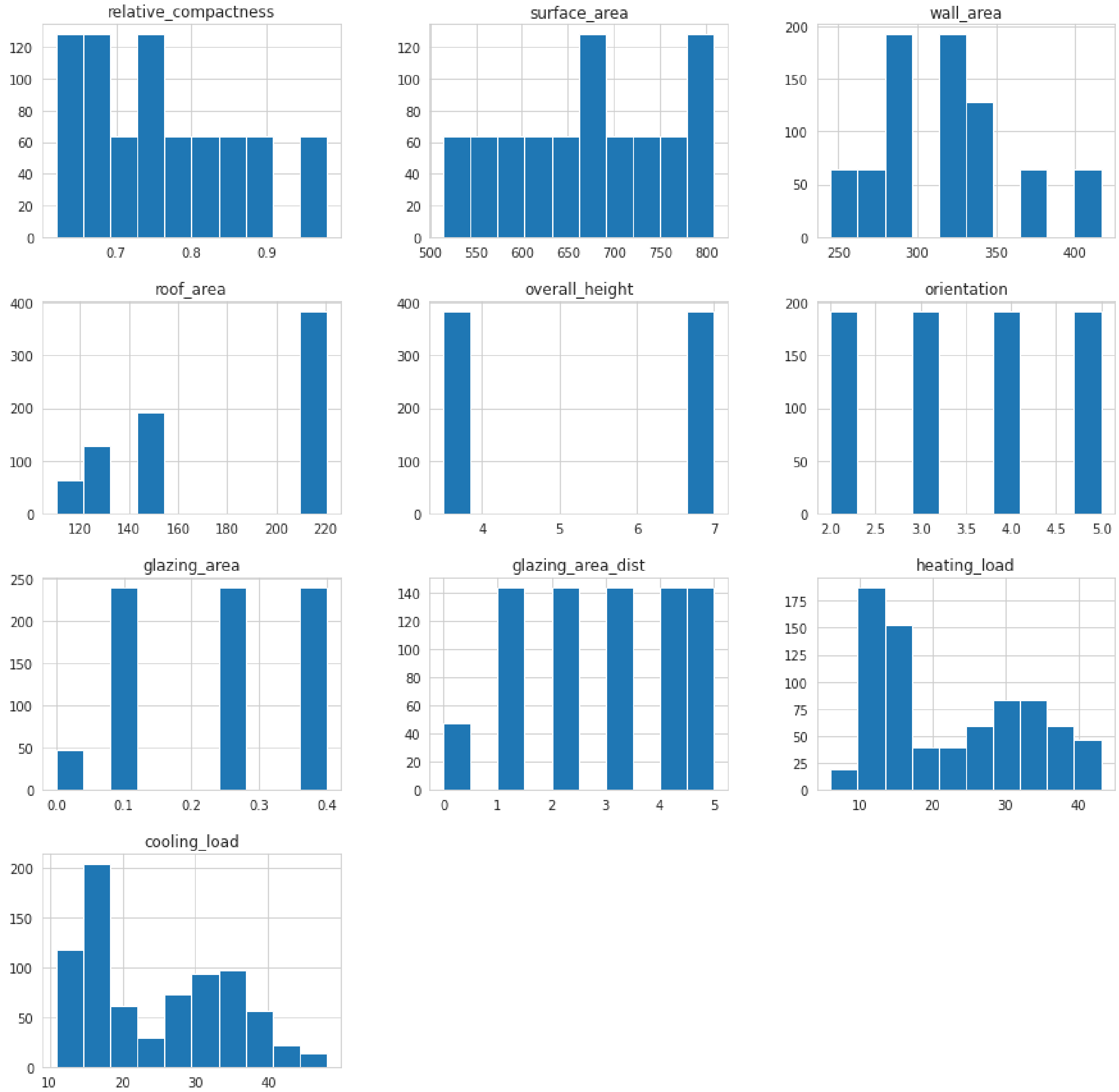
```
df2 = pd.read_excel('ENB2012_data.xlsx')
```

```
df2.columns=["relative_compactness","surface_area","wall_area","roof_area","overall_height","orientation",
                   "glazing_area","glazing_area_dist","heating_load","cooling_load"]
df2
```

	relative_compactness	surface_area	wall_area	roof_area	overall_height	orientation	glazing_area	glazing_area_dist	hea
0	0.98	514.50	294.00	110.25	7.00	2	0.00	0	
1	0.98	514.50	294.00	110.25	7.00	3	0.00	0	
2	0.98	514.50	294.00	110.25	7.00	4	0.00	0	
3	0.98	514.50	294.00	110.25	7.00	5	0.00	0	
4	0.90	563.50	318.50	122.50	7.00	2	0.00	0	
...	...	...	...	...	...	...	...	...	
763	0.64	784.00	343.00	220.50	3.50	5	0.40	5	
764	0.62	808.50	367.50	220.50	3.50	2	0.40	5	
765	0.62	808.50	367.50	220.50	3.50	3	0.40	5	
766	0.62	808.50	367.50	220.50	3.50	4	0.40	5	
767	0.62	808.50	367.50	220.50	3.50	5	0.40	5	

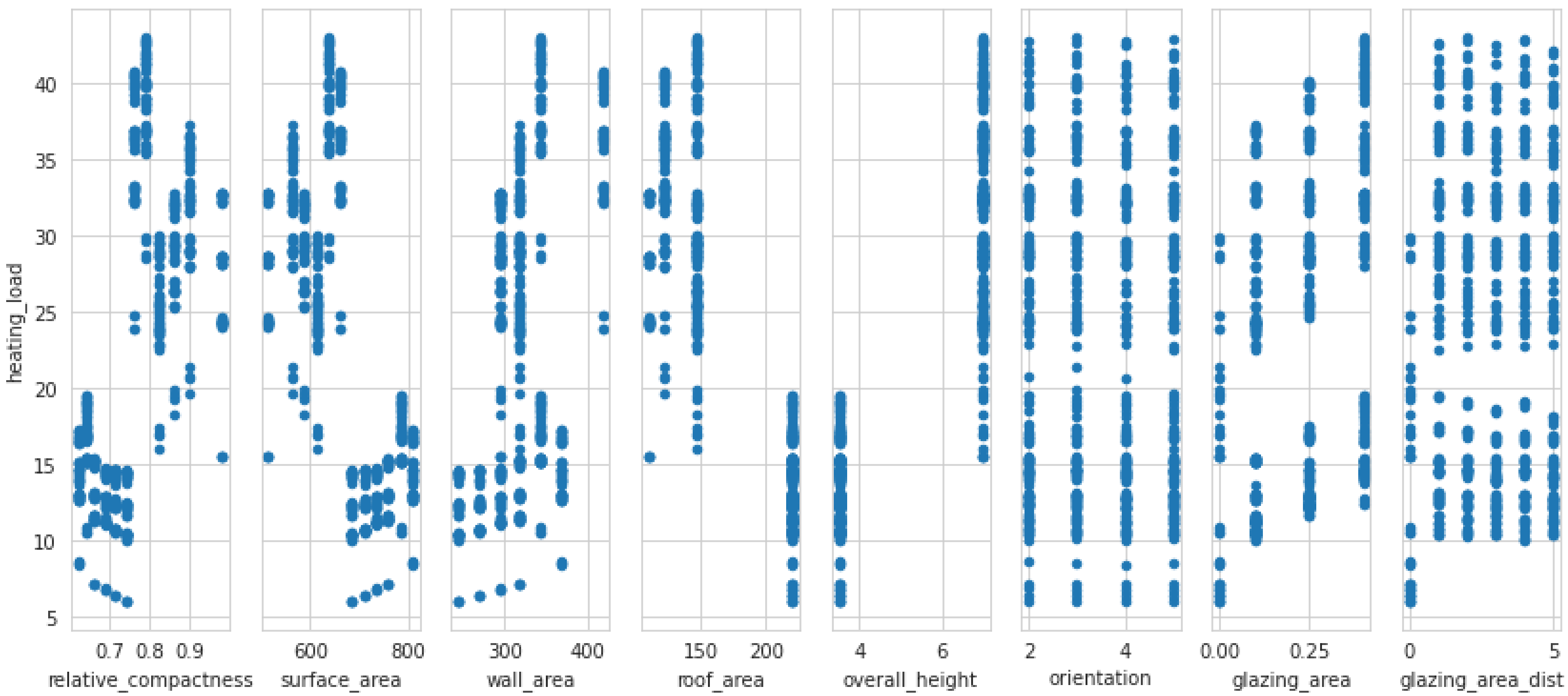
768 rows × 10 columns

```
df2.hist(figsize=(15,15))
plt.show()
```


relative_compactness
surface_area
wall_area
roof_area
overall_height
orientation
glazing_area
glazing_area_dist
heating_load
cooling_load

```
fig, axs = plt.subplots(1, 8, sharey=True)
df2.plot(kind='scatter', x='relative_compactness', y='heating_load', ax=axs[0], figsize=(14, 6))
df2.plot(kind='scatter', x='surface_area', y='heating_load', ax=axs[1])
df2.plot(kind='scatter', x='wall_area', y='heating_load', ax=axs[2])
df2.plot(kind='scatter', x='roof_area', y='heating_load', ax=axs[3])
df2.plot(kind='scatter', x='overall_height', y='heating_load', ax=axs[4])
df2.plot(kind='scatter', x='orientation', y='heating_load', ax=axs[5])
df2.plot(kind='scatter', x='glazing_area', y='heating_load', ax=axs[6])
df2.plot(kind='scatter', x='glazing_area_dist', y='heating_load', ax=axs[7])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fab9f82a4d0>
```




```
df2.isnull().sum()
```

```
relative_compactness    0
surface_area            0
wall_area               0
roof_area               0
overall_height          0
orientation             0
glazing_area            0
glazing_area_dist       0
heating_load            0
```

```
cooling_load            0
dtype: int64
```

df2.info

```
<bound method DataFrame.info of      relative_compactness  surface_area  wall_area  roof_area  overall_height  \
0                    0.98         514.5      294.0     110.25             7.0
1                    0.98         514.5      294.0     110.25             7.0
2                    0.98         514.5      294.0     110.25             7.0
3                    0.98         514.5      294.0     110.25             7.0
4                    0.90         563.5      318.5     122.50             7.0
..                    ...           ...        ...        ...             ...
763                  0.64         784.0      343.0     220.50             3.5
764                  0.62         808.5      367.5     220.50             3.5
765                  0.62         808.5      367.5     220.50             3.5
766                  0.62         808.5      367.5     220.50             3.5
767                  0.62         808.5      367.5     220.50             3.5

     orientation  glazing_area  glazing_area_dist  heating_load  cooling_load
0              2           0.0                  0         15.55         21.33
1              3           0.0                  0         15.55         21.33
2              4           0.0                  0         15.55         21.33
3              5           0.0                  0         15.55         21.33
4              2           0.0                  0         20.84         28.28
..           ...           ...                ...           ...           ...
763            5           0.4                  5         17.88         21.40
764            2           0.4                  5         16.54         16.88
765            3           0.4                  5         16.44         17.11
766            4           0.4                  5         16.48         16.61
767            5           0.4                  5         16.64         16.03

[768 rows x 10 columns]>
```

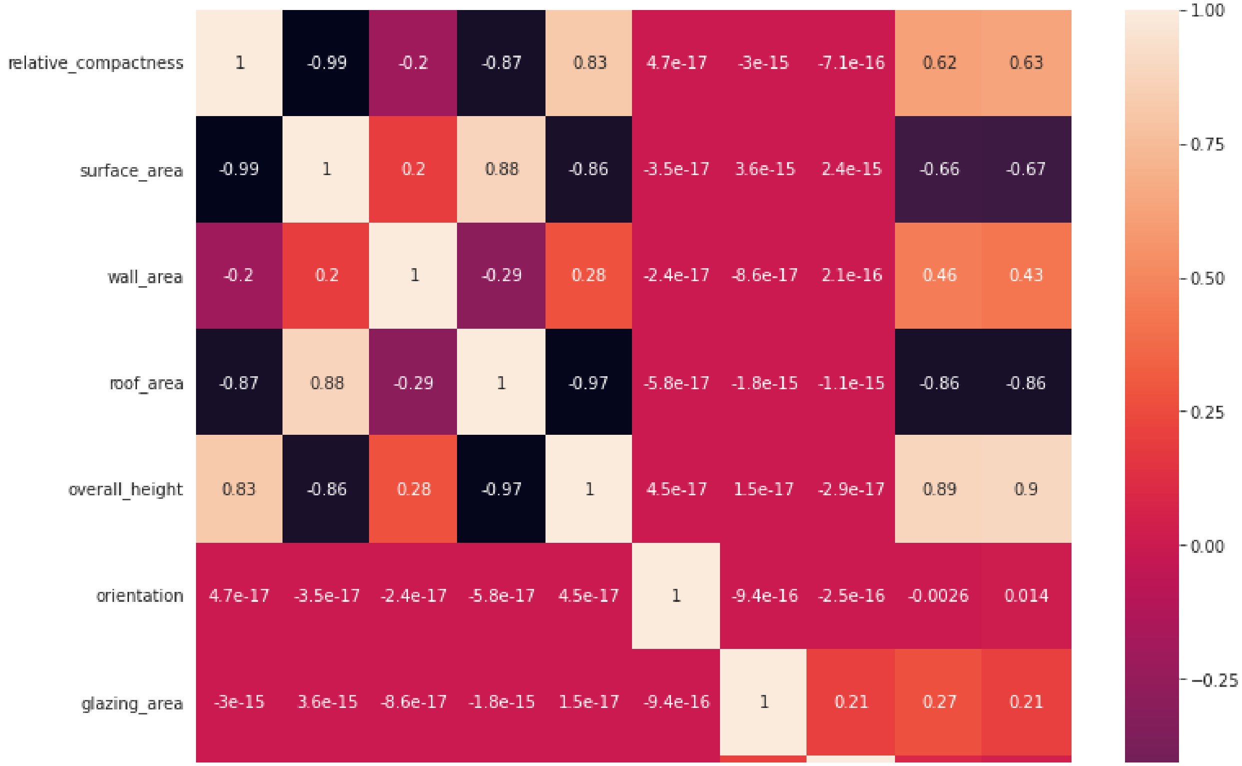
df2.describe

```
<bound method NDFrame.describe of      relative_compactness  surface_area  wall_area  roof_area  overall_height  \
0                    0.98         514.5      294.0     110.25             7.0
1                    0.98         514.5      294.0     110.25             7.0
2                    0.98         514.5      294.0     110.25             7.0
3                    0.98         514.5      294.0     110.25             7.0
4                    0.90         563.5      318.5     122.50             7.0
..                    ...           ...        ...        ...             ...
763                  0.64         784.0      343.0     220.50             3.5
764                  0.62         808.5      367.5     220.50             3.5
```

```
765                   0.62         808.5      367.5     220.50              3.5
766                   0.62         808.5      367.5     220.50              3.5
767                   0.62         808.5      367.5     220.50              3.5

     orientation  glazing_area  glazing_area_dist  heating_load  cooling_load
0              2           0.0                  0         15.55         21.33
1              3           0.0                  0         15.55         21.33
2              4           0.0                  0         15.55         21.33
3              5           0.0                  0         15.55         21.33
4              2           0.0                  0         20.84         28.28
..           ...           ...                ...           ...           ...
763            5           0.4                  5         17.88         21.40
764            2           0.4                  5         16.54         16.88
765            3           0.4                  5         16.44         17.11
766            4           0.4                  5         16.48         16.61
767            5           0.4                  5         16.64         16.03

[768 rows x 10 columns]>
```

```
plt.figure(figsize=(12,12))
sns.heatmap(df2.corr(),annot=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7faba034d610>




```
pd.set_option('display.float_format',lambda x: '{:,.2f}'.format(x) if abs(x) < 10000 else '{:,.0f}'.format(x))
df2.corr()
```

	relative_compactness	surface_area	wall_area	roof_area	overall_height	orientation	glazing_area	glazin
relative_compactness	1.00	-0.99	-0.20	-0.87	0.83	0.00	-0.00	
surface_area	-0.99	1.00	0.20	0.88	-0.86	-0.00	0.00	

```
#Normalize inputs, set output
from sklearn.preprocessing import Normalizer
nr = Normalizer(copy=False)

X = df2.drop(['heating_load','cooling_load'], axis=1)
X = nr.fit_transform(X)
y = df2[['heating_load','cooling_load']]
```

```
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_state = 123)
```

```
X_new = pd.DataFrame({'relative_compactness': [df2.relative_compactness.min(), df2.relative_compactness.max()]})
X_new.head()
```

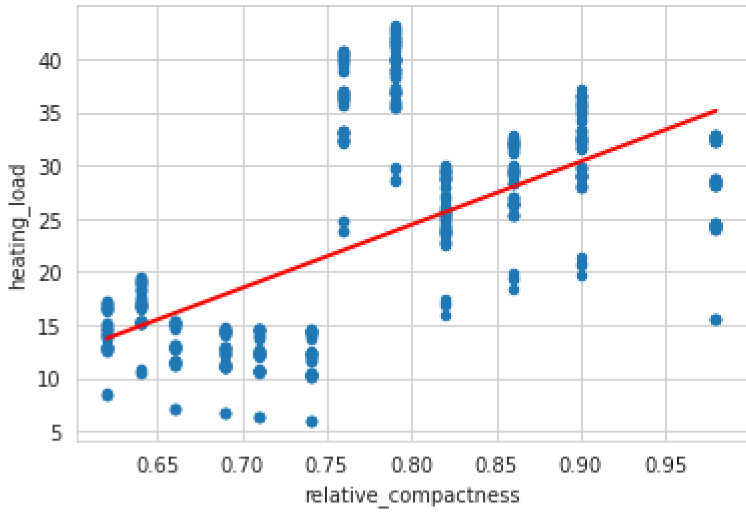
	relative_compactness
0	0.62
1	0.98

```
import statsmodels.formula.api as smf

# Create fitted model.
lm = smf.ols(formula='heating_load ~ relative_compactness', data=df2).fit()

# Get coefficients.
lm.params

pred = lm.predict(X_new)
pred
```

```
0   13.75
1   35.12
dtype: float64
```

```
df2.plot(kind='scatter', x='relative_compactness', y='heating_load')
plt.plot(X_new, pred, c='red', linewidth=2)
```

```
[<matplotlib.lines.Line2D at 0x7faba03bd590>]
```



Implementation of Lasso Regression

```
from sklearn.linear_model import Lasso
from sklearn.linear_model import LassoCV
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
lasso_model = Lasso().fit(X_train, y_train)
lasso_model
```

```
Lasso()
```

```
lasso_model.intercept_
```

```
array([22.101875  , 24.35225694])
```

```
lasso_model.coef_
```

```
array([[ 0., -0.,  0., -0.,  0.,  0.,  0.,  0.],
```

```
        [ 0., -0.,  0., -0.,  0.,  0.,  0.,  0.]])
```

```
from sklearn.metrics import mean_squared_error

y_pred = lasso_model.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

```
    10.000941383873894
```

```
lasso_model.score(X,y)
```

```
    -0.0005141088871829513
```

Implementation of Ridge Regression

```
from sklearn.linear_model import Ridge
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 42) # Train test split
ridge_model = Ridge(alpha = 5).fit(X_train, y_train)
ridge_model
```

```
    Ridge(alpha=5)
```

```
ridge_model.intercept_
```

```
    array([22.32902789, 24.52907455])
```

```
ridge_model.coef_
```

```
    array([[  0.11629168, -11.54544662,  36.83527657, -24.19036159,
              1.88144148,   0.24205811,   0.07361704,   0.41832675],
           [  0.11410307, -10.71744662,  34.30447853, -22.51096257,
              1.81488642,   0.2554495 ,   0.05716537,   0.31520485]])
```

```
ridge_model = Ridge().fit(X_train, y_train)
y_pred = ridge_model.predict(X_train)
RMSE = np.sqrt(mean_squared_error(y_train, y_pred))
RMSE
```

```
    5.314821337489512
```

```
y_pred = ridge_model.predict(X_test)
RMSE = np.sqrt(mean_squared_error(y_test, y_pred))
RMSE
```

```
5.328363319249427
```

```
ridge_model.score(X,y)
```

```
0.7049581597106621
```

✓ 21s completed at 6:21 PM