Final Project
COMP.4220 Machine Learning
UMass Lowell, Spring 2022
Khim Sok
April 29, 2022

I. Introduction

Throughout this course, we have learned some of the useful machine learning techniques to be used in both classification and regression problem. In this paper, we analyze credit card fraud by using Logistic Regression, Support Vector Machine, Decision Tree, Random Forest, and Deep Learning Neural Network on Credit Card Fraud Detection Data Set. In this case, we report accuracies of 97.41%, 99.69%, 98.27%, 99.95%, 99.10%, respectively. Finally, the Regression Problem, we analyzed energy efficiency of heating load and cooling by using Linear Regression, Polynomial Regression, Regularized Regression with Lasso and Ridge, and Support Vector Regression. These models report mean square error (MSE) values, [7.88, 9.70], [0.24, 2.51], [22.03, 21.66], [2.91, 3.22], [0.04, 0.08] respectively.

II. Classification Problem: Credit Card Fraud Detection Data Set

A. The Data Set

In the Classification problem, we take a look into a dataset of credit card transactions that were made in September 2013 by European cardholders.

These transactions occurred within two days, where it found 492 fraudulent transactions out of 284,807 transactions. This dataset is significant imbalanced, the positive class indicates fraudulent which reports 0.172% of all transaction, and zero class indicates non-fraudulent.

This dataset was collected and analyzed during a research collaboration of Worldline and the Machine Learning Group of ULB (Université Libre de Bruxelles) on big data mining and fraud detection.

B. Data Exploration

The dataset contains 30 features (V1, V2, …V28), where each feature indicates the principal components obtained with PCA, due to confidential issue, the features are used V(n) instead. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. Feature 'Amount' is the transaction Amount. Feature 'Class' is the response variable that take value 1 in case of fraud and 0 for non-fraud.

We defined V1, V2, …, V28 as X, and Class as Y which divided into two classes, 1 and 0. Below is the histogram of Feature 'Class' which report 492 fraud transaction out of 284315 transactions.

```
Total number of legit transaction:  284315
Total number of fraud transaction:  492
Fraud percentage: 0.1727485630620034%
```

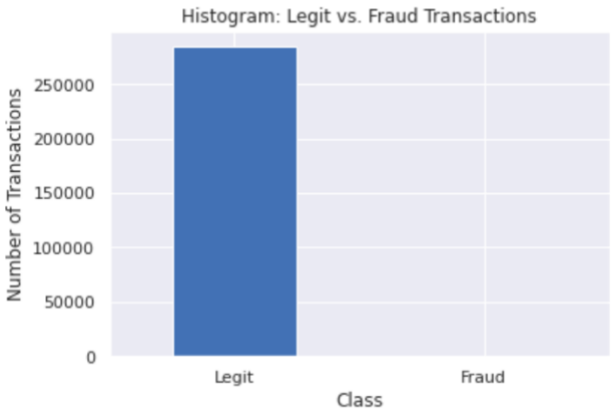


Figure 1. Histogram of Credit Card Data Set

In addition, heat map can help us understand the correlation between each feature to show us the positive vs. negative value in Feature 'Class' that correlated to each feature. Notices that V7 and V20 have slightly darker green which indicates higher positive correlation with feature 'Amount'.

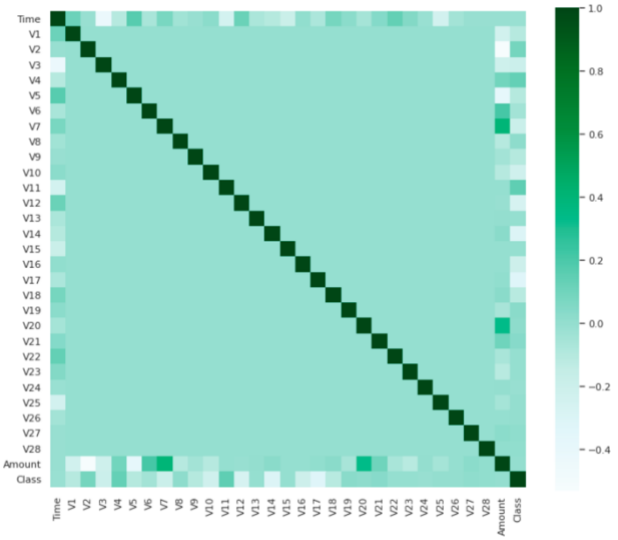


Figure 2. Correlation Heatmap of Credit Card Fraud Detection

C. *Data Pre-Processing*

Before applying any machine learning technique, we first divide the data into training set and testing with radio of 7:3 at the random state of 420. We then apply StandardScaler of Scikit-learn on both training and testing set to standardizes features by removing the mean and scaling to unit variance.

D. *Address the Imbalanced Data Set*

As can be seen in the Figure1, the dataset is significantly unbalanced, therefore, we need to preprocess the data to make it balanced and give us a more accurate result by using SMOTE algorithms from Imbalanced-learn package.

E. *Logistic Regression*

Logistic Regression, the very first technique we used to fit the training set, it is a process of modeling the probability of discrete outcome given an input variable. Since this model is very simple, it takes a very short time to make a prediction. Result of the model reports a 97.42% accuracy, with 0.0571 precision score, 0.9236 of recall score, F1 score of 0.1075, and AUC score of 0.9745.

F. *Support Vector Machine*

Since the logistic regression a relatively low precision score, we use the second method which is Support Vector Machine (SVM). This model is used to define the data point between classifications. SVM is more advance and complex that take a longer time to execute and fit the training set, in this case, it took us about 20 minutes long to complete the execution. The model reports 99.67% accuracy, with precision score of 0.3262, recall score of 0.7431, and with F1 score of 0.4534.

G. *Decision Tree*

Although the precision score of SVM model is larger compared to Logistic Regression model, it is still under performance. So, we pick up another method, Decision Tree. This model also great for classification problem. As the result, it reports 98.27% accuracy, precision score of 0.0786, recall score of 0.8611, with the F1 score of 0.1440, and AUC score of 0.9535.

H. *Random Forest*

At this point, we are still not satisfied with the results of Decision Tress since it reported a significant low precision score. Therefore, we decided to continue with another technique which is Random Forest. The model reports 99.96% accuracy, precision score of 0.9037, recall score of 0.8472, with F1 score of 0.8745, and AUC score of 0.9879.

I. *Deep Learning: Neural Network*

Although the Random Forest provide a high-performance prediction on every score, we want to try another advance technique which is Deep Learning (Neural Network), to see if it gives us a better result. We add one input layer, three hidden layers, and one output layer. As the result, we get 99.11% accuracy, precision score of 0.1474, recall score of 0.8958, and the F1 score of 0.2531.

J. *Analysis*

Begin with Logistic Regression, we get a very low precision score which was clearly not a good result. Second, Support Vector Machine reports a below 50% of precision score, which is not good. Third, Decision Tree, also report relatively low precision score. Fourth, Random Forest, report an overall great score compared to previous methods. Finally, Neural Network, also report low precision score.

Overall, we found that the Random Forest model provides the best result of prediction with relatively high score of 99.59% accuracy, along with high precision score of 90%, and recall score of 84% which is higher compared to other methods we used. It also gives us highest F1 score and AUC score of 0.8745 and 0.9878 respectively.

TABLE 1
CREDIT CARD FRAUD DETECTION DATA SET CLASSIFICATION REPORT

	Accuracy	Precision	Recall	F1-score
Logistic Regression	0.9741	0.057	0.9236	0.1075
Support Vector Machine	0.9969	0.3262	0.7431	0.4534
Decision Tree	0.9827	0.0786	0.8611	0.1440
Random Forest	0.9996	0.9037	0.8472	0.8745
Neural Network	0.9911	0.1474	0.8958	0.2531

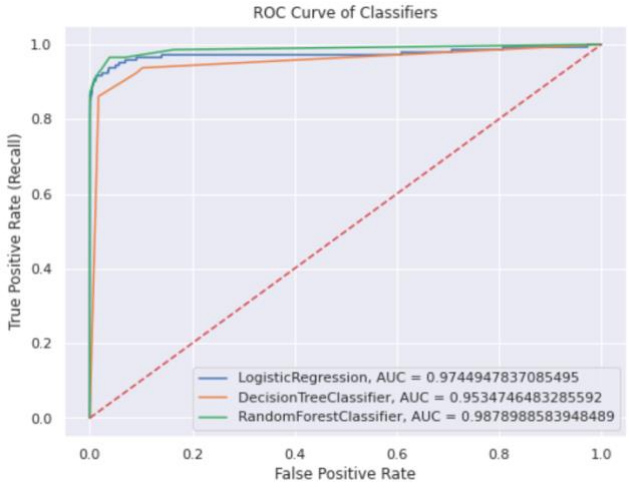


Figure 3. ROC Curve of different Classifiers showing AUC score comparison

III. REGRESSION PROBLEM: ENERGY EFFICIENCY DATA SET

A. *The Data Set*

According to the source, this data set provides an access to data of energy efficiency of heating load and cooling load requirements of building as a function of building parameters Energy analysis using 12 different building orientation and shape. The data collected by University of California, Irvine (UC Irvine).

B. *Data Exploration*

The dataset contains ten features, includes eight input variables which are building orientations, and two output variables which are heating load and cooling load.

As the dataset provided with a simple variable name, 'X1, X2, …, X8' and 'Y1 and Y2', we need to renamed the features based on the definition provided in the source website, as 'Relative Compactness, Surface Area, Wall Area, Roof Area, Overall Height, Orientation, Glazing Area, Glazing Area Distribution, Heating Load, Cooling Load' respectively.

We plot each feature of the dataset to help us understand better the distribution. We can see the different variant between each feature in the figure below.

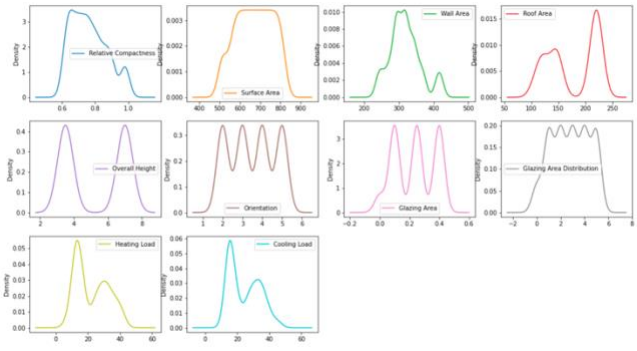


Figure 4. Plots of Energy Efficiency Data Set Feature Distribution

We also use heatmap for better understanding on the data correlation. Base on the heatmap generated from the dataset, it tells us that the Overall Height, Wall Area, and Relative Compactness have the most influent on the result of Heating Load and Cooling Load.

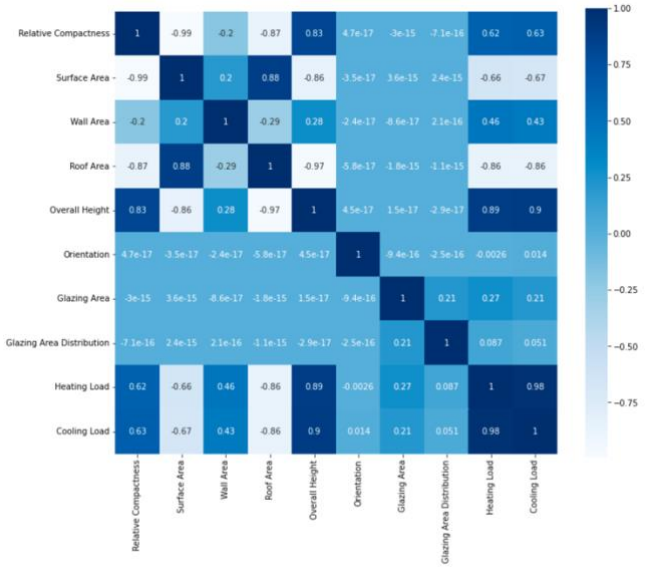


Figure 5. Correlation Heatmap of Energy Efficiency Data Set

C. *Data Pre-Processing*

Prior to applying any regression methods on the dataset, we need to split them into training set and testing set with radio of 7:3 and random state of 420. We also apply StandardScaler of Scikit-learn.

D. *Linear Regression*

The first method we use in this regression problem is Linear Regression model. It reports margin square error (MSE) of 7.8809, explain score of 0.9285 for Heating Load while Cooling Load reported MSE of 9.7031 and explained score of 0.8992. The MSE evaluation score for both heating and cooling load is relatively high as we want the MSE as small as possible which should be very closed to zero.

```
Heating Load
Evaluation MSE:  7.880991665622867
Explained score: 0.9285009975188163
```

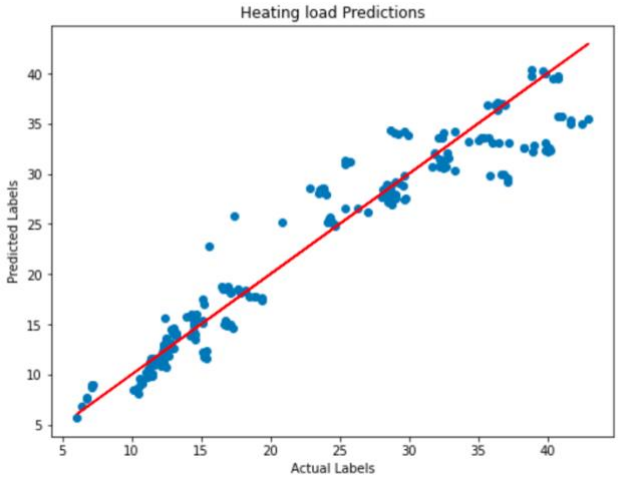


Figure 6.1. Linear Regression on Heating Load

```
Cooling Load
Evaluation MSE:  9.703113537912834
Explained score: 0.8991828201441125
```

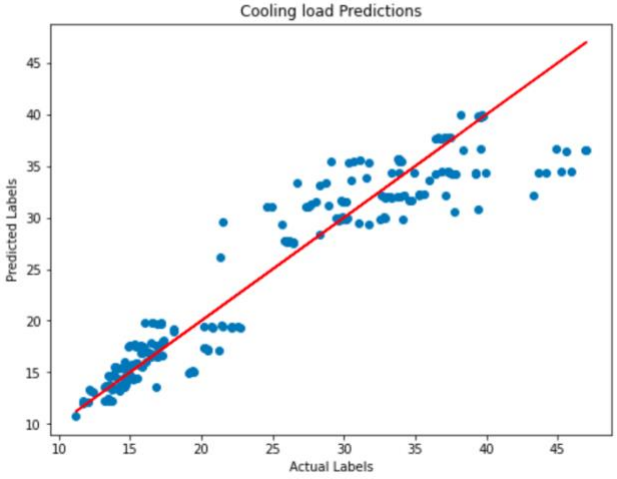


Figure 6.2. Linear Regression on Cooling Load

E. *Polynomial Regression*

The second method we applied on the data set is Polynomial Regression. It reports MSE of 0.2424 and explained score of 0.9977 for Heating Load while the Cooling Load reported MSE of 2.5081 and explained score of 0.9736.

```
Heating Load
Evaluation MSE:  0.24245235038691273
Explained score: 0.9977711687193728
```

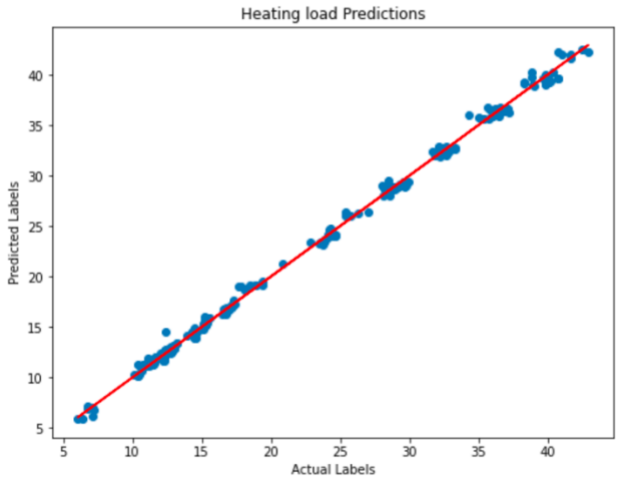


Figure 7.1. Polynomial Regression on Heating Load

```
Cooling Load
Evaluation MSE:  2.5081070275579176
Explained score: 0.9736643895147131
```

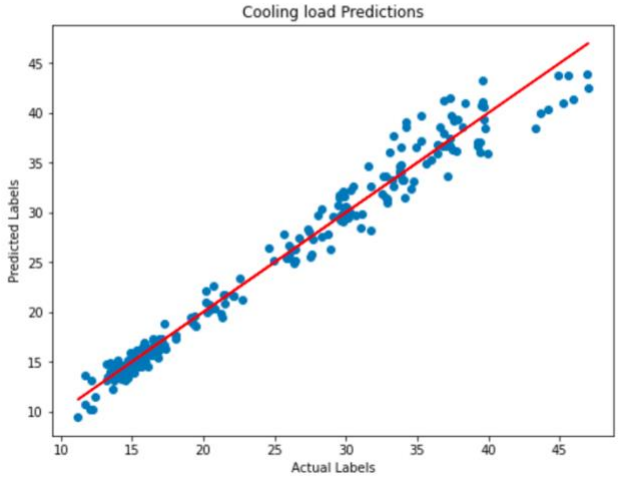


Figure 7.2. Polynomial Regression on Cooling Load

F. Regularized Regression

Although the polynomial regression reported a good result, but we want to try another method which is Regularized Regression. This method has two main functions which are Lasso Regression, and Ridge Regression. We applied both functions to see which one has better performance on prediction.

F.1. LASSO Regression

With the Lasso Regression, we received a report of MSE 22.0372 and explained score 0.7950 for Heating Load. On the other hand, the Cooling Load reported MSE of 21.6648 and explained score of 0.7716. The MSE is too high compared to any other regression. Let's take a look into Ridge Regression.

```
Lasso Heating Load
Lasso Test MSE: 22.037265260760975
Explained score: 0.795019984290958
```

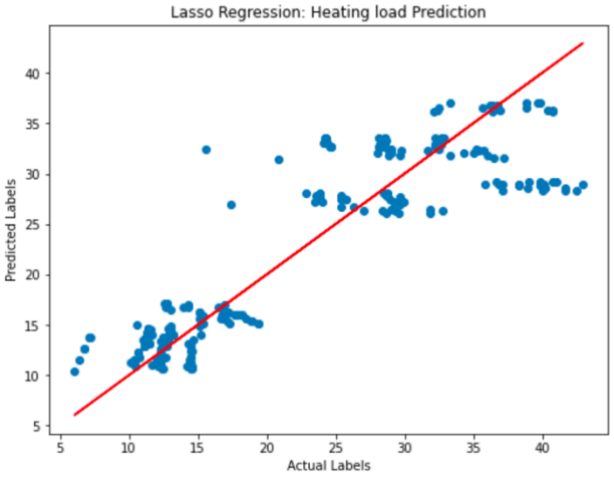


Figure 8.1. LASSO Regression on Heating Load

```
Lasso Heating Load
Lasso Test MSE: 21.66489307653214
Explained score: 0.7716161573512936
```

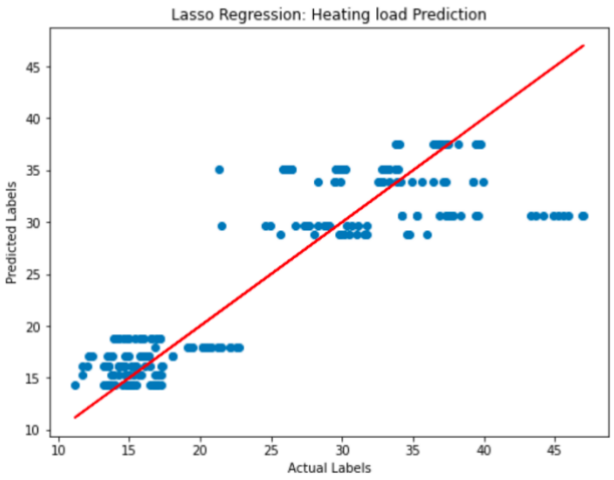


Figure 8.2. LASSO Regression on Cooling Load

F.2. Ridge Regression

The Ridge Regression reports MSE of 2.9146 and explained score of 0.9225 for Heating Load while Cooling Load reported MSE of 3.2165 and explained score of 0.8922. These results are a lot better than Lasso Regression.

```
Heating Load
Ridge Test MSE: 2.9146763715821744
Explained score: 0.9225404933129574
```

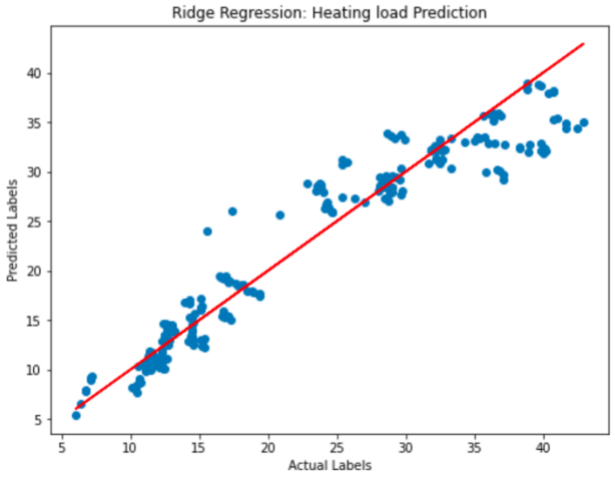


Figure 9.1. Ridge Regression on Heating Load

```
Cooling Load
Ridge Test MSE: 3.2164805690505323
Explained score: 0.8921572470259151
```

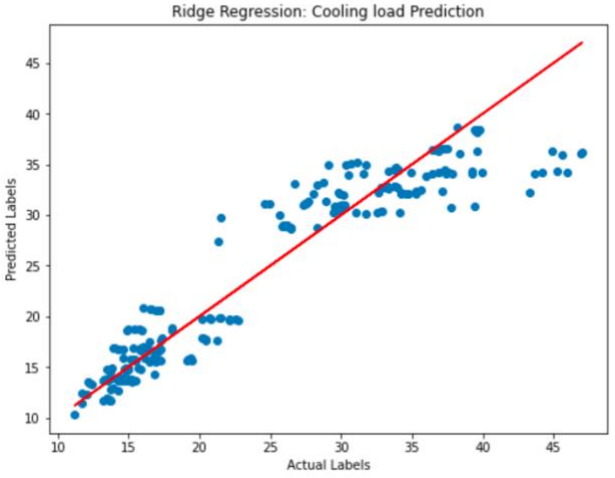


Figure 9.2. Ridge Regression on Cooling Load

G. Support Vector Regression

We still not satisfied with the regression we used so far, so we try another regression which is Support Vector Regression (SVR). It reports MSE of 0.0493 and explained score of 0.9551 for Heating Load while Cooling Load received MSE of 0.0893 and explained score of 0.9169. These results are the best performance so far.

```
SVR Heating Load
Evaluation MSE:  0.04928249856721011
Explained score: 0.9550994798153676
```

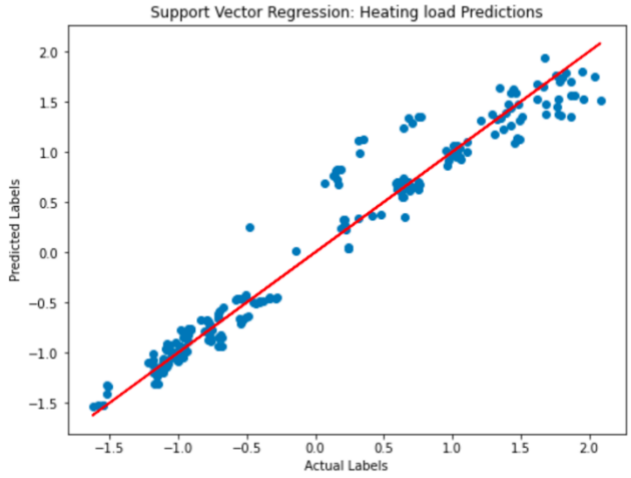


Figure 10.1. SVR on Heating Load

```
SVR Cooling Load
Evaluation MSE:  0.08931678352443816
Explained score: 0.9169573680386633
```

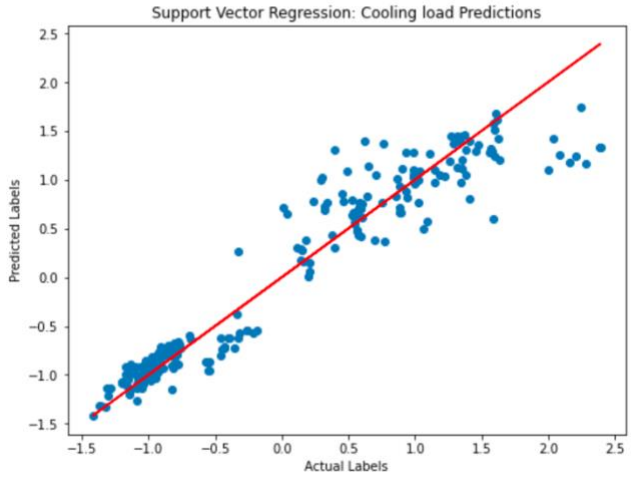


Figure 10.2. SVR on Cooling Load

H. Analysis

Begin with Linear Regression, though it is simple and easy to use, it did not perform well in term of MSE for both Heating Load and Cooling Load. Polynomial Regression on the other hand, perform better than Linear Regression but still has a MSE relatively larger than what we want. LASSO Regression, not to mentioned, has the worst performance that reports really large MSE. In contrast, Ridge Regression perform better than LASSO Regression but still not the best. Finally, we got SVR which report the best performance among all regression that you used which can be explained in Table 2 below.

TABLE 2
ENERGY EFFICIENCY DATA SET REGRESSION REPORT

	Heating Load		Cooling Load	
	MSE	Explained	MSE	Explained
Linear Regression	7.8809	0.9285	9.7031	0.8991
Polynomial Regression	0.2424	0.9977	2.5081	0.9736
LASSO Regression	22.0372	0.7950	21.6648	0.7716
Ridge Regression	2.9146	0.9225	3.2164	0.8921
Support Vector Regression	0.0492	0.9551	0.0893	0.9169

IV. CONCLUSION

After completed this project, we have developed our understanding on variety of different methods and techniques that we can be used on both Classification Problem and Regression Problem. We have learned that a single problem can have different approach to solve the problem, and also to make sure we get the best performance, it requires multiple training with various methods, and analyze the accuracy score, precision score, recall score, F1 score, AUC score, as well as Margin Square Error to understand and obtain the best solution among many other solutions.

This course work provides us a great fundamental knowledge in Machine Learning which can be applied to our future career and other projects.

Problem I: Classicfication - Credit Card Fraud Detection

Data Exploration

In [53]:
```
import numpy as np
import pandas as pd

creditcard_fraud_df = pd.read_csv('creditcard.csv')
creditcard_fraud_df
```

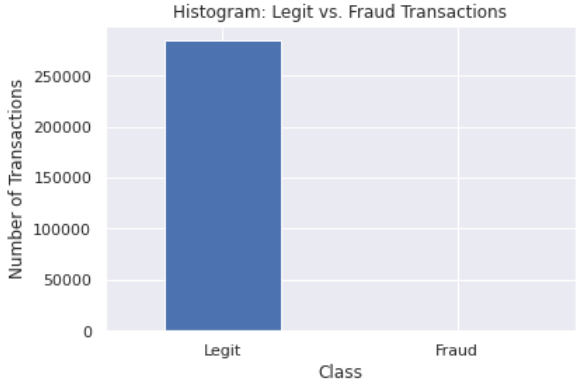
Out[53]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.111864	1.014
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.924384	0.012
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.578229	-0.037
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.800049	-0.163
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.643078	0.376

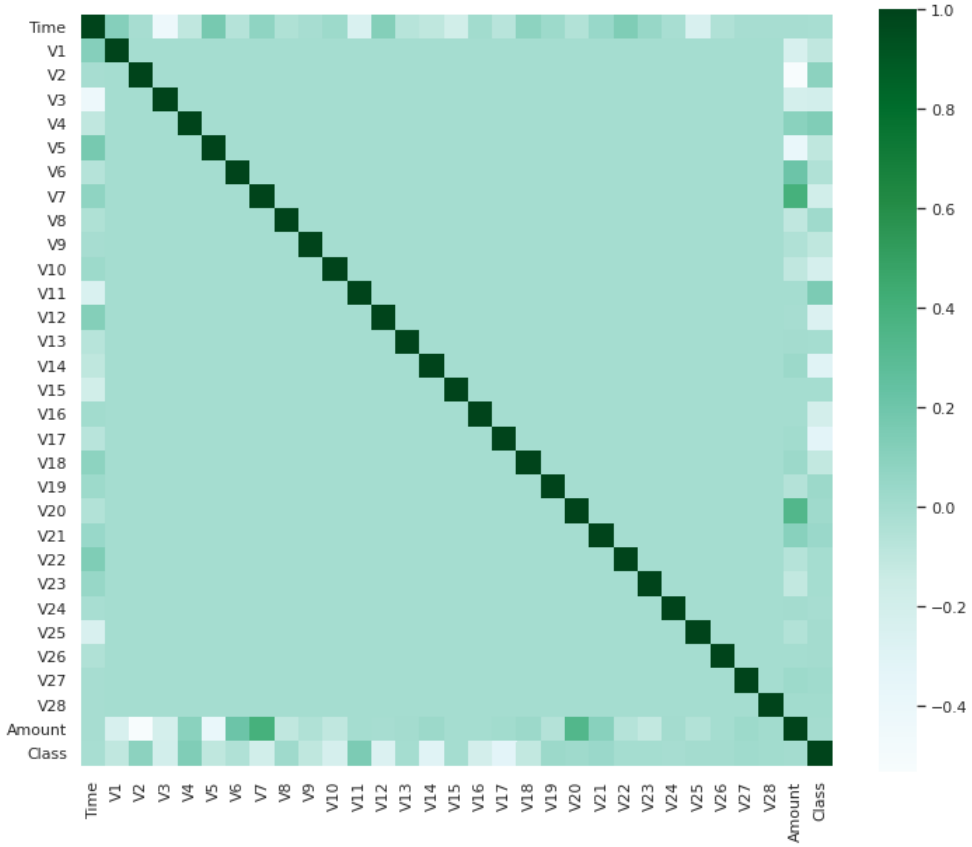
284807 rows × 31 columns

In [54]:
```
import matplotlib.pyplot as plt

class_count = creditcard_fraud_df.value_counts(creditcard_fraud_df['Class'])
class_count.plot(kind='bar', rot=0)
plt.xticks(range(2), ['Legit', 'Fraud'])
plt.ylabel('Number of Transactions')
plt.title('Histogram: Legit vs. Fraud Transactions')


legit = len(creditcard_fraud_df[creditcard_fraud_df.Class==0])
fraud = len(creditcard_fraud_df[creditcard_fraud_df.Class==1])

print("Total number of legit transaction: ", legit)
print("Total number of fraud transaction: ", fraud)
print("Fraud percentage: {}%".format(fraud/(fraud+legit)*100))

plt.show()
```

```
Total number of legit transaction:  284315
Total number of fraud transaction:  492
Fraud percentage: 0.1727485630620034%
```



In [55]:
```
import seaborn as sns
import matplotlib.pyplot as plt

corrmat = creditcard_fraud_df.corr()
plt.figure(figsize=(12,10))
sns.heatmap(corrmat, square=True, cmap='BuGn', linecolor='w', annot=False)
plt.show()
```



Preprocessing Data Set

In [56]:
```
creditcard_fraud_df.dropna()

# Data Matrix X
X = np.array(creditcard_fraud_df.iloc[:,1:30])

# Target y
y = np.array(creditcard_fraud_df.iloc[:,-1])

print(X.shape, y.shape)
```

```
(284807, 29) (284807,)
```

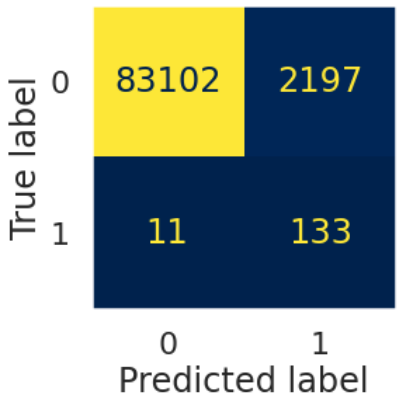
In [57]:
```
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=420)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Instantiate the classfiers and make a list
classifiers = []
```

Case1: Logistic Regression With Data-Level Preprocessing

In [58]:
```
from sklearn.utils import multiclass
from sklearn.linear_model import LogisticRegression

# Train
clf_log_reg = LogisticRegression(solver='lbfgs', multi_class='ovr', max_iter=1000)

classifiers.append(clf_log_reg)

clf_log_reg.fit(X_train_resampled, y_train_resampled)

# Test
y_pred = clf_log_reg.predict(X_test)
y_prob = clf_log_reg.predict_proba(X_test)[::,1]
```

In [59]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score

sns.set_context("poster")

# Create fusion matrix
disp = plot_confusion_matrix(clf_log_reg, X_test, y_test, cmap='cividis', colorbar=False)

# Find Precision and Recall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

plt.grid(False)
plt.show()

# Print Precsion and Recall
print("Precision: {}, Recall: {}".format(precision, recall))
```

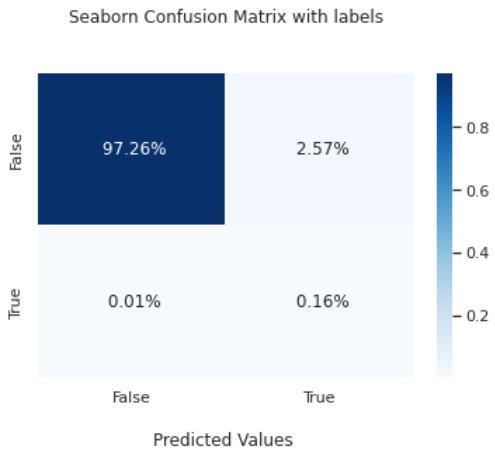
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusio
n_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use
one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
Precision: 0.05708154506437768, Recall: 0.9236111111111112
```

In [60]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling

mat = confusion_matrix(y_test, y_pred)
print(mat)
ax = sns. heatmap (mat/np. sum(mat), annot=True, fmt=' .2%',cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n') ;
ax.set_xlabel('\nPredicted Values ')
ax.set_label('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True' ])
## Display the visualization of the Confusion Matrix.
plt.show()
```

```
[[83102  2197]
 [   11   133]]
```



Evaluate Performance of Logistics Regression with Data-level Preprocessing

In [61]:
```
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import roc_auc_score

print('Logistics Regression')
print('Evaluation:\t{}'.format(mse(y_test, y_pred, squared=False)))
print('Accuracy: \t{}'.format(accuracy_score(y_test, y_pred)))
print('Precision: \t{}'.format(precision_score(y_test, y_pred)))
print('Recall: \t{}'.format(recall_score(y_test, y_pred)))
print('F1 Score: \t{}'.format(f1_score(y_test, y_pred)))
print('AUC Score: \t{}'.format(roc_auc_score(y_test, y_prob)))
```

```
Logistics Regression
Evaluation:     0.16075381570282382
Accuracy:       0.9741582107369825
Precision:      0.05708154506437768
Recall:         0.9236111111111112
F1 Score:       0.10751818916734034
AUC Score:      0.9744947837085495
```

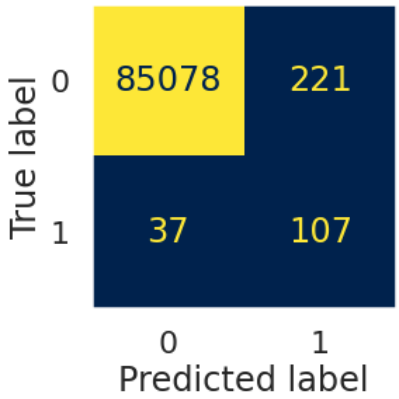
Case 2: Support Vector Machine

In [62]:
```
from sklearn.svm import SVC

svm_clf = SVC(gamma='auto')

svm_clf.fit(X_train_resampled, y_train_resampled)

y_pred = svm_clf.predict(X_test)

confusion_matrix(y_test, y_pred)
```

Out[62]:
```
array([[85078,   221],
       [   37,   107]])
```

In [63]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score

sns.set_context("poster")

# Create fusion matrix
disp = plot_confusion_matrix(svm_clf, X_test, y_test, cmap='cividis', colorbar=False)

# Find Precision and Recall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

plt.grid(False)
plt.show()

# Print Precsion and Recall
print("Precision: {}, Recall: {}".format(precision, recall))
```

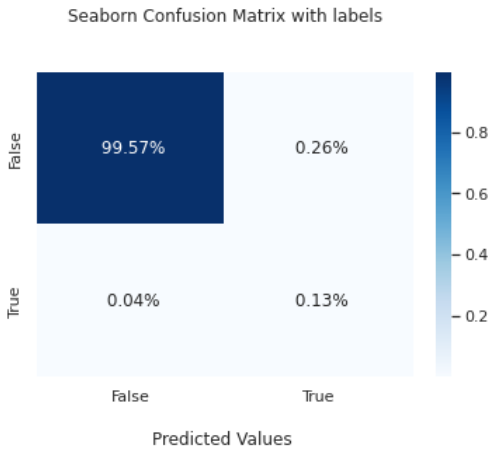
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusio
n_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use
one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
Precision: 0.32621951219512196, Recall: 0.7430555555555556
```

In [64]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling

mat = confusion_matrix(y_test, y_pred)
print(mat)
ax = sns. heatmap (mat/np. sum(mat), annot=True, fmt=' .2%',cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n') ;
ax.set_xlabel('\nPredicted Values ')
ax.set_label('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True' ])
## Display the visualization of the Confusion Matrix.
plt.show()
```

```
[[85078   221]
 [   37   107]]
```



In [65]:
```
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import roc_auc_score

print('Support Vector Machine with SVC')
print('Evaluation:\t{}'.format(mse(y_test, y_pred, squared=False)))
print('Accuracy: \t{}'.format(accuracy_score(y_test, y_pred)))
print('Precision: \t{}'.format(precision_score(y_test, y_pred)))
print('Recall: \t{}'.format(recall_score(y_test, y_pred)))
print('F1 Score: \t{}'.format(f1_score(y_test, y_pred)))
```

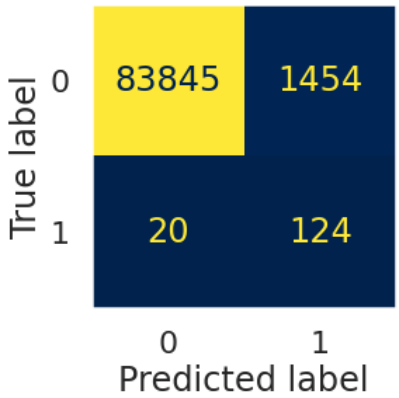
```
Support Vector Machine with SVC
Evaluation:     0.054950494971208796
Accuracy:       0.9969804431024192
Precision:      0.32621951219512196
Recall:         0.7430555555555556
F1 Score:       0.45338983050847464
```

Case 3: Decision Tree

In [66]:
```
from sklearn.tree import DecisionTreeClassifier

tree_clf = DecisionTreeClassifier(max_depth=2, random_state=42)

classifiers.append(tree_clf)

tree_clf.fit(X_train_resampled, y_train_resampled)

y_pred = tree_clf.predict(X_test)
y_prob = tree_clf.predict_proba(X_test)[::,1]
```

In [67]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score

sns.set_context("poster")

# Create fusion matrix
disp = plot_confusion_matrix(tree_clf, X_test, y_test, cmap='cividis', colorbar=False)

# Find Precision and Recall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

plt.grid(False)
plt.show()

# Print Precsion and Recall
print("Precision: {}, Recall: {}".format(precision, recall))
```

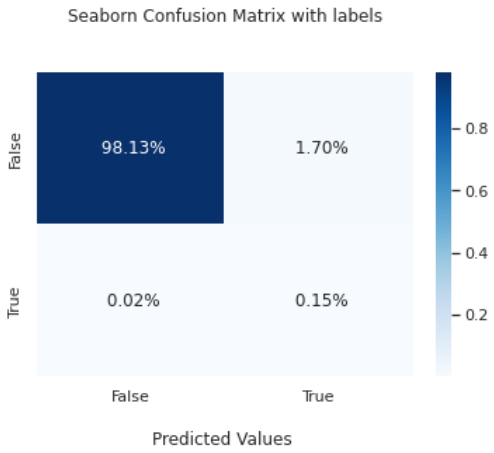
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusio
n_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use
one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
Precision: 0.07858048162230671, Recall: 0.8611111111111112
```

In [68]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling

mat = confusion_matrix(y_test, y_pred)
print(mat)
ax = sns. heatmap (mat/np. sum(mat), annot=True, fmt=' .2%',cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n') ;
ax.set_xlabel('\nPredicted Values ')
ax.set_label('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True' ])
## Display the visualization of the Confusion Matrix.
plt.show()
```

```
[[83845  1454]
 [   20   124]]
```



In [69]:
```
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import roc_auc_score

print('Decision Tree')
print('Evaluation:\t{}'.format(mse(y_test, y_pred, squared=False)))
print('Accuracy: \t{}'.format(accuracy_score(y_test, y_pred)))
print('Precision: \t{}'.format(precision_score(y_test, y_pred)))
print('Recall: \t{}'.format(recall_score(y_test, y_pred)))
print('F1 Score: \t{}'.format(f1_score(y_test, y_pred)))
print('AUC Score: \t{}'.format(roc_auc_score(y_test, y_prob)))
```

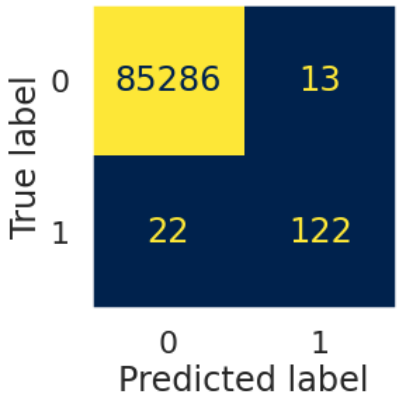
```
Decision Tree
Evaluation:     0.13134407838379697
Accuracy:       0.982748733073511
Precision:      0.07858048162230671
Recall:         0.8611111111111112
F1 Score:       0.1440185830429733
AUC Score:      0.9534746483285592
```

Case 4: Random Forest

In [70]:
```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=123, random_state=123)

classifiers.append(rnd_clf)

rnd_clf.fit(X_train_resampled, y_train_resampled)

y_pred = rnd_clf.predict(X_test)
y_prob = rnd_clf.predict_proba(X_test)[::,1]
```

In [71]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score

sns.set_context("poster")

# Create fusion matrix
disp = plot_confusion_matrix(rnd_clf, X_test, y_test, cmap='cividis', colorbar=False)

# Find Precision and Recall
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)

plt.grid(False)
plt.show()

# Print Precsion and Recall
print("Precision: {}, Recall: {}".format(precision, recall))
```

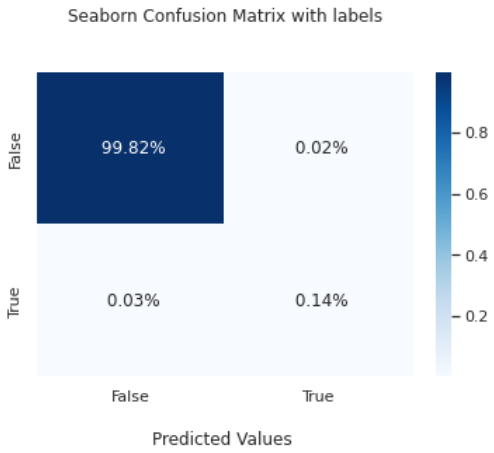
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function plot_confusio
n_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use
one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




```
Precision: 0.9037037037037037, Recall: 0.8472222222222222
```

In [72]:
```
import matplotlib.pyplot as plt
import seaborn as sns;
import seaborn as sns; sns.set() # for plot styling

mat = confusion_matrix(y_test, y_pred)
print(mat)
ax = sns. heatmap (mat/np. sum(mat), annot=True, fmt=' .2%',cmap='Blues')

ax.set_title('Seaborn Confusion Matrix with labels\n\n') ;
ax.set_xlabel('\nPredicted Values ')
ax.set_label('Actual Values ');

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['False', 'True'])
ax.yaxis.set_ticklabels(['False', 'True' ])
## Display the visualization of the Confusion Matrix.
plt.show()
```

```
[[85286    13]
 [   22   122]]
```

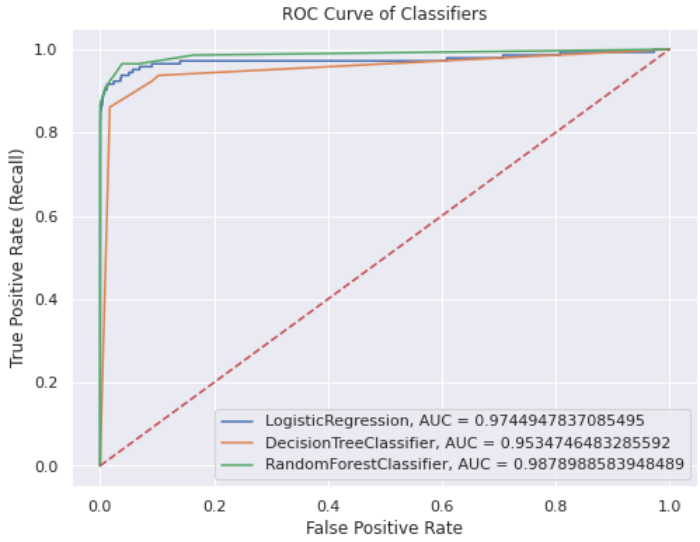


In [73]:
```
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import roc_auc_score

print('Random Forest')
print('Evaluation:\t{}'.format(mse(y_test, y_pred, squared=False)))
print('Accuracy: \t{}'.format(accuracy_score(y_test, y_pred)))
print('Precision: \t{}'.format(precision_score(y_test, y_pred)))
print('Recall: \t{}'.format(recall_score(y_test, y_pred)))
print('F1 Score: \t{}'.format(f1_score(y_test, y_pred)))
print('AUC Score: \t{}'.format(roc_auc_score(y_test, y_prob)))
```

```
Random Forest
Evaluation:     0.02023931351818346
Accuracy:       0.9995903701883126
Precision:      0.9037037037037037
Recall:         0.8472222222222222
F1 Score:       0.8745519713261649
AUC Score:      0.9878988583948489
```

Summary ROC and AUC Scores

In [74]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 6))

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(X_train_resampled, y_train_resampled)
    y_prob = model.predict_proba(X_test)[::,1]

    fpr, tpr, _ = roc_curve(y_test,  y_prob)
    auc = roc_auc_score(y_test, y_prob)
    plt.plot(fpr, tpr, label="{}, AUC = {}".format(cls.__class__.__name__, auc))
    plt.legend(loc="lower right")

plt.title("ROC Curve of Classifiers")
plt.plot([0,1], [0,1], '--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate (Recall)")
plt.grid("True")
plt.show()
```



Case 5: Neural Network Models

In [75]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense

nn_clf = Sequential()

nn_clf.add(keras.layers.Flatten(input_shape=X.shape[1:]))
nn_clf.add(keras.layers.Dense(units=6, kernel_initializer='uniform', activation='relu', input_dim=11))
nn_clf.add(keras.layers.Dense(units=6, kernel_initializer='uniform', activation='relu'))
nn_clf.add(keras.layers.Dense(units=1, kernel_initializer='uniform', activation='sigmoid'))
nn_clf.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [76]:
```
batch_size = 5
epochs = 20
history = nn_clf.fit(X_train_resampled, y_train_resampled, epochs=epochs, batch_size=batch_size)
```

```
Epoch 1/20
79607/79607 [==============================] - 143s 2ms/step - loss: 0.0641 - accuracy: 0.9746
Epoch 2/20
79607/79607 [==============================] - 137s 2ms/step - loss: 0.0369 - accuracy: 0.9883
Epoch 3/20
79607/79607 [==============================] - 137s 2ms/step - loss: 0.0303 - accuracy: 0.9908
Epoch 4/20
79607/79607 [==============================] - 140s 2ms/step - loss: 0.0273 - accuracy: 0.9919
Epoch 5/20
79607/79607 [==============================] - 143s 2ms/step - loss: 0.0255 - accuracy: 0.9927
Epoch 6/20
79607/79607 [==============================] - 139s 2ms/step - loss: 0.0233 - accuracy: 0.9935
Epoch 7/20
79607/79607 [==============================] - 143s 2ms/step - loss: 0.0224 - accuracy: 0.9939
Epoch 8/20
79607/79607 [==============================] - 142s 2ms/step - loss: 0.0219 - accuracy: 0.9941
Epoch 9/20
79607/79607 [==============================] - 143s 2ms/step - loss: 0.0214 - accuracy: 0.9943
Epoch 10/20
79607/79607 [==============================] - 146s 2ms/step - loss: 0.0205 - accuracy: 0.9946
Epoch 11/20
79607/79607 [==============================] - 138s 2ms/step - loss: 0.0204 - accuracy: 0.9947
Epoch 12/20
79607/79607 [==============================] - 138s 2ms/step - loss: 0.0204 - accuracy: 0.9946
Epoch 13/20
79607/79607 [==============================] - 138s 2ms/step - loss: 0.0195 - accuracy: 0.9950
Epoch 14/20
79607/79607 [==============================] - 146s 2ms/step - loss: 0.0191 - accuracy: 0.9950
Epoch 15/20
79607/79607 [==============================] - 141s 2ms/step - loss: 0.0187 - accuracy: 0.9953
Epoch 16/20
79607/79607 [==============================] - 144s 2ms/step - loss: 0.0183 - accuracy: 0.9953
Epoch 17/20
79607/79607 [==============================] - 141s 2ms/step - loss: 0.0179 - accuracy: 0.9955
Epoch 18/20
79607/79607 [==============================] - 137s 2ms/step - loss: 0.0180 - accuracy: 0.9955
Epoch 19/20
79607/79607 [==============================] - 138s 2ms/step - loss: 0.0176 - accuracy: 0.9955
Epoch 20/20
79607/79607 [==============================] - 156s 2ms/step - loss: 0.0173 - accuracy: 0.9956
```

In [77]:
```
y_pred = nn_clf.predict(X_test)
y_pred = (y_pred>0.5)
print(y_pred)

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[False]
 [False]
 [False]
 ...
 [False]
 [False]
 [False]]
[[84553   746]
 [   15   129]]
```

In [79]:
```
from sklearn.metrics import confusion_matrix, precision_score, accuracy_score, f1_score, recall_score
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import roc_auc_score

print('Evaluation:\t{}'.format(mse(y_test, y_pred, squared=False)))
print('Accuracy: \t{}'.format(accuracy_score(y_test, y_pred)))
print('Precision: \t{}'.format(precision_score(y_test, y_pred)))
print('Recall: \t{}'.format(recall_score(y_test, y_pred)))
print('F1 Score: \t{}'.format(f1_score(y_test, y_pred)))
```

```
Evaluation:     0.09437437404811198
Accuracy:       0.991093477523027
Precision:      0.14742857142857144
Recall:         0.8958333333333334
F1 Score:       0.253189401373896
```

Problem II: Regression Problem - Energy Efficiency

Import Data

In [2]:
```
import pandas as pd
ee_df = pd.read_excel('ENB2012_data.xlsx')
ee_df
```

Out[2]:

	X1	X2	X3	X4	X5	X6	X7	X8	Y1	Y2
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28
...	...	...	...	...	...	...	...	...	...	...
763	0.64	784.0	343.0	220.50	3.5	5	0.4	5	17.88	21.40
764	0.62	808.5	367.5	220.50	3.5	2	0.4	5	16.54	16.88
765	0.62	808.5	367.5	220.50	3.5	3	0.4	5	16.44	17.11
766	0.62	808.5	367.5	220.50	3.5	4	0.4	5	16.48	16.61
767	0.62	808.5	367.5	220.50	3.5	5	0.4	5	16.64	16.03

768 rows × 10 columns

In [3]:
```
ee_df = ee_df.rename(columns={'X1':'Relative Compactness',
                              'X2':'Surface Area',
                              'X3':'Wall Area',
                              'X4':'Roof Area',
                              'X5':'Overall Height',
                              'X6':'Orientation',
                              'X7':'Glazing Area',
                              'X8':'Glazing Area Distribution',
                              'Y1':'Heating Load',
                              'Y2':'Cooling Load'})
ee_df.head()
```

Out[3]:

	Relative Compactness	Surface Area	Wall Area	Roof Area	Overall Height	Orientation	Glazing Area	Glazing Area Distribution	Heating Load	Cooling Load
0	0.98	514.5	294.0	110.25	7.0	2	0.0	0	15.55	21.33
1	0.98	514.5	294.0	110.25	7.0	3	0.0	0	15.55	21.33
2	0.98	514.5	294.0	110.25	7.0	4	0.0	0	15.55	21.33
3	0.98	514.5	294.0	110.25	7.0	5	0.0	0	15.55	21.33
4	0.90	563.5	318.5	122.50	7.0	2	0.0	0	20.84	28.28

In [4]:
```
ee_df.describe()
```

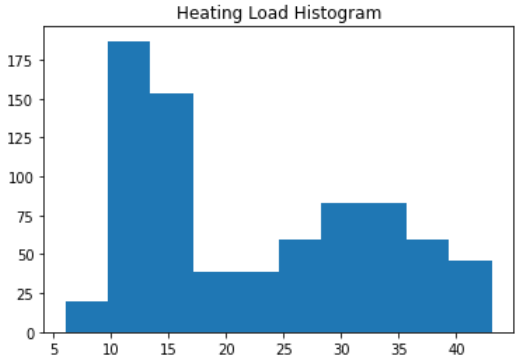
Out[4]:

	Relative Compactness	Surface Area	Wall Area	Roof Area	Overall Height	Orientation	Glazing Area	Glazing Area Distribution	Heating Load	Cooling Load
count	768.000000	768.000000	768.000000	768.000000	768.00000	768.000000	768.000000	768.00000	768.000000	768.000000
mean	0.764167	671.708333	318.500000	176.604167	5.25000	3.500000	0.234375	2.81250	22.307195	24.587760
std	0.105777	88.086116	43.626481	45.165950	1.75114	1.118763	0.133221	1.55096	10.090204	9.513306
min	0.620000	514.500000	245.000000	110.250000	3.50000	2.000000	0.000000	0.00000	6.010000	10.900000
25%	0.682500	606.375000	294.000000	140.875000	3.50000	2.750000	0.100000	1.75000	12.992500	15.620000
50%	0.750000	673.750000	318.500000	183.750000	5.25000	3.500000	0.250000	3.00000	18.950000	22.080000
75%	0.830000	741.125000	343.000000	220.500000	7.00000	4.250000	0.400000	4.00000	31.667500	33.132500
max	0.980000	808.500000	416.500000	220.500000	7.00000	5.000000	0.400000	5.00000	43.100000	48.030000

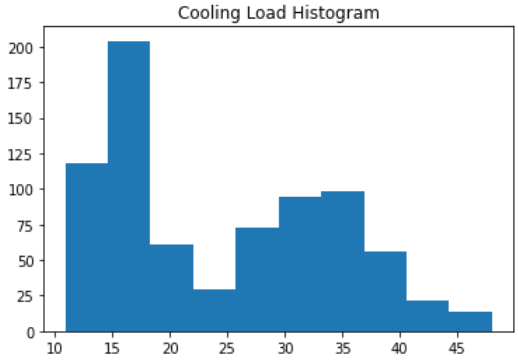
In [5]:
```
print(ee_df.isnull().sum())
```

```
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
```

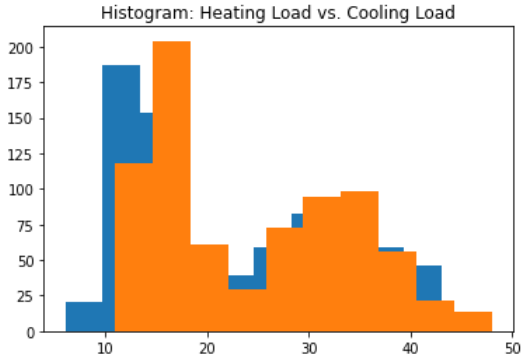
In [6]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(6,4))
plt.hist(ee_df['Heating Load'])
plt.title("Heating Load Histogram")
plt.show()
```



In [7]:
```
import matplotlib.pyplot as plt
plt.hist(ee_df['Cooling Load'])
plt.title("Cooling Load Histogram")
plt.show()
```

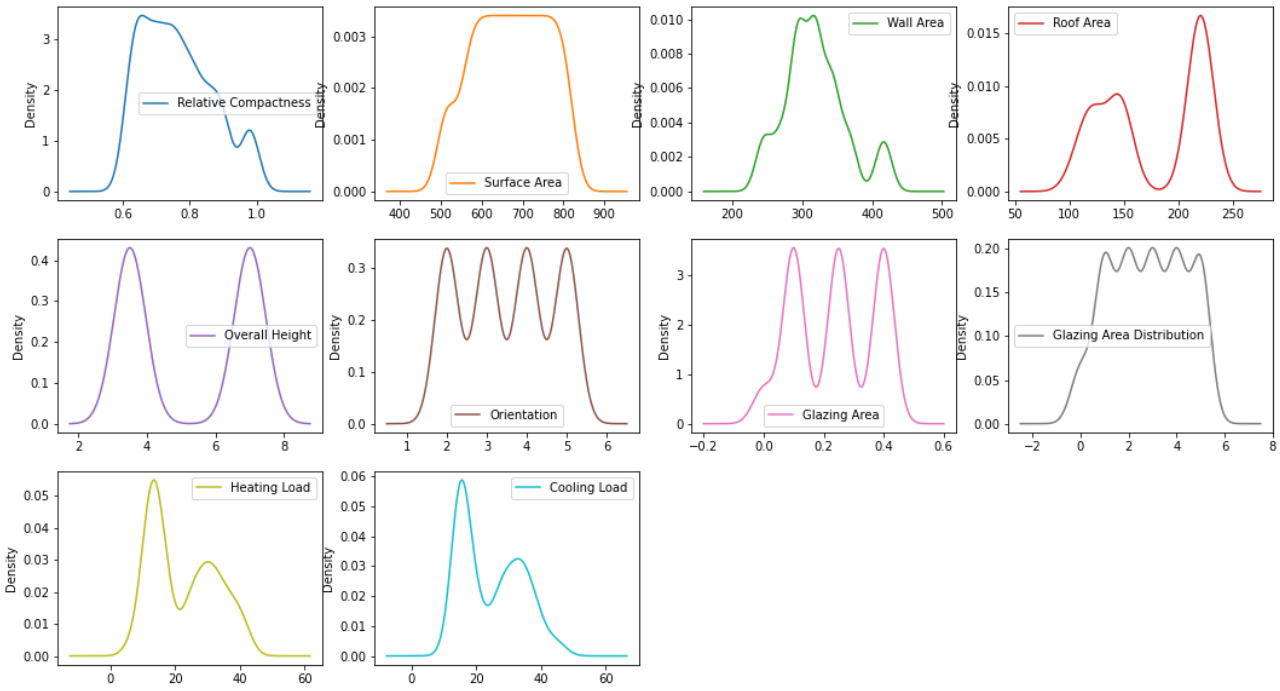


In [8]:
```
import matplotlib.pyplot as plt
plt.hist(ee_df['Heating Load'])
plt.hist(ee_df['Cooling Load'])
plt.title("Histogram: Heating Load vs. Cooling Load")
plt.show()
```

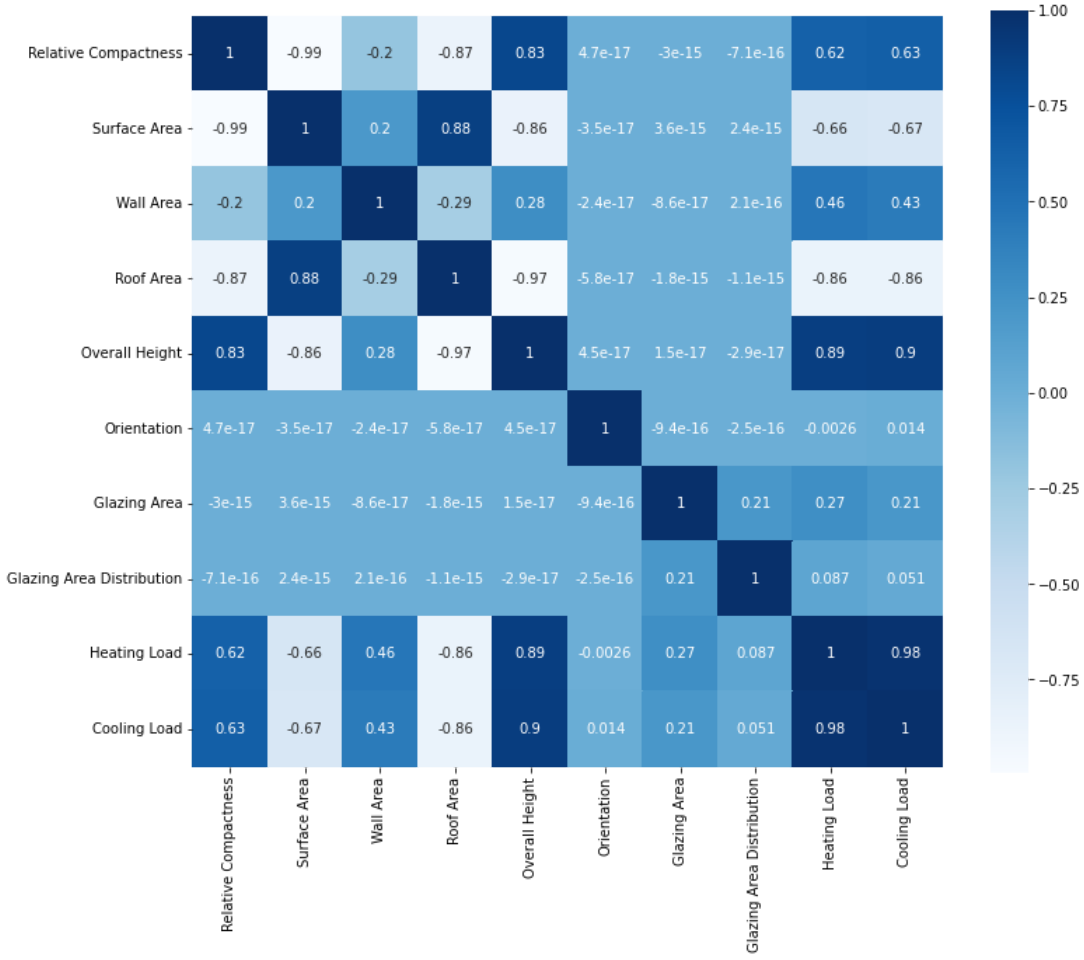


In [9]:
```
from IPython.core.pylabtools import figsize
import matplotlib.pyplot as plt
ee_df.plot(kind='density', subplots=True, layout=(3,4), sharex=False, sharey=False, figsize=(18,10))
```

Out[9]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fd1541e00d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd1542146d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd1541cacd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd15418d310>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fd154143910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd1540fbf10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd1540bd5d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd154073b10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fd154073b50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd154038290>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd153fa6d90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fd153f6b3d0>]],
      dtype=object)
```



In [10]:
```
import seaborn as sns
import matplotlib.pyplot as plt
corrmat = ee_df.corr()
plt.figure(figsize=(12,10))
sns.heatmap(corrmat, square=True, cmap='Blues', linecolor='w', annot=True)
plt.show()
```



Define X and Y

In [29]:
```
import numpy as np

X = np.array(ee_df.iloc[:,:8])
y = ee_df.iloc[:,-2:]
y1 = np.array(y.iloc[:,:1])
y2 = np.array(y.iloc[:,1:])

print(X.shape, y.shape)
print(y1.shape, y2.shape)
```

```
(768, 8) (768, 2)
(768, 1) (768, 1)
```

In [30]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train1, y_test1, y_train2, y_test2 = train_test_split(X, y1, y2, test_size=0.3, random_stat
e=420)

print(X_train.shape, X_test.shape)
print(y_train1.shape, y_test1.shape)
print(y_train2.shape, y_test2.shape)
```

```
(537, 8) (231, 8)
(537, 1) (231, 1)
(537, 1) (231, 1)
```

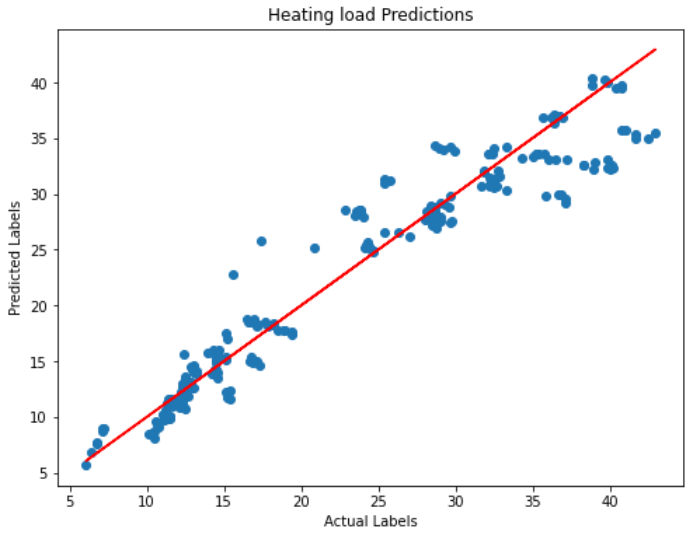
Case 1: Linear Regression

In [31]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Define pip
pipe1 = Pipeline([
                ('scaler', StandardScaler()),
                ('regressor', LinearRegression()),
])

pipe2 = Pipeline([
                ('scaler', StandardScaler()),
                ('regressor', LinearRegression())
])


# Train
pipe1.fit(X_train, y_train1)
pipe2.fit(X_train, y_train2)


# Predict
y_pred1 = pipe1.predict(X_test)
y_pred2 = pipe2.predict(X_test)

print(y_pred1.shape, y_pred2.shape)
```

```
(231, 1) (231, 1)
```

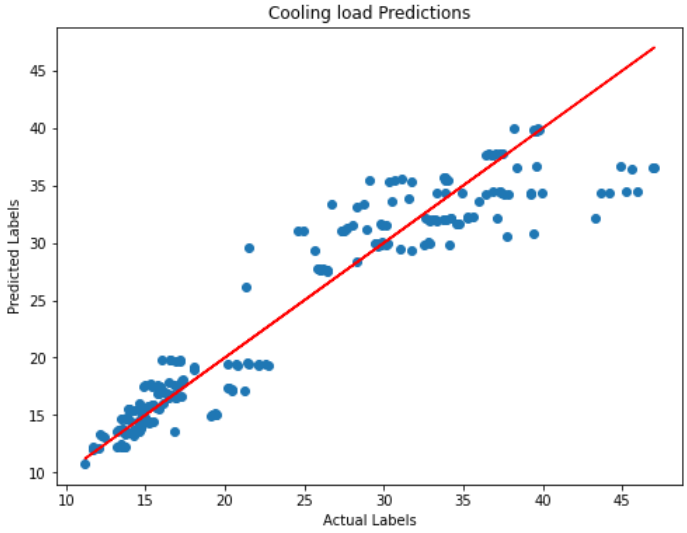
In [32]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score
print('Heating Load')
print('Evaluation MSE:\t {}'.format(mse(y_test1, y_pred1, squared='False')))
print('Explained score: {}'.format(explained_variance_score(y_test1, y_pred1)))

plt.figure(figsize=(8,6))
plt.scatter(y_test1, y_pred1)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Heating load Predictions')

plt.plot(y_test1, y_test1, color='red')
plt.show()
```

```
Heating Load
Evaluation MSE:  7.880991665622867
Explained score: 0.9285009975188163
```



In [33]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score
print('Cooling Load')
print('Evaluation MSE:\t {}'.format(mse(y_test2, y_pred2, squared='False')))
print('Explained score: {}'.format(explained_variance_score(y_test2, y_pred2)))

plt.figure(figsize=(8,6))
plt.scatter(y_test2, y_pred2)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Cooling load Predictions')

plt.plot(y_test2, y_test2, color='red')
plt.show()
```

```
Cooling Load
Evaluation MSE:  9.703113537912834
Explained score: 0.8991828201441125
```

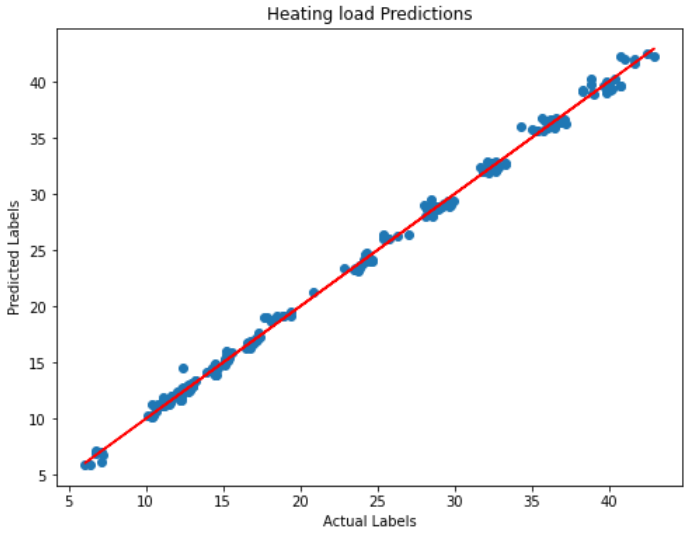


Case 2: Polynomial Regression

In [34]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import PolynomialFeatures

# Define pip
pipe1 = Pipeline([
                ('scaler', StandardScaler()),
                ('poly', PolynomialFeatures(degree=3, include_bias='False')),
                ('regressor', LinearRegression())
])

pipe2 = Pipeline([
                ('scaler', StandardScaler()),
                ('poly', PolynomialFeatures(degree=3, include_bias='False')),
                ('regressor', LinearRegression()),
])


# Train
pipe1.fit(X_train, y_train1)
pipe2.fit(X_train, y_train2)


# Predict
y_pred1 = pipe1.predict(X_test)
y_pred2 = pipe2.predict(X_test)
```

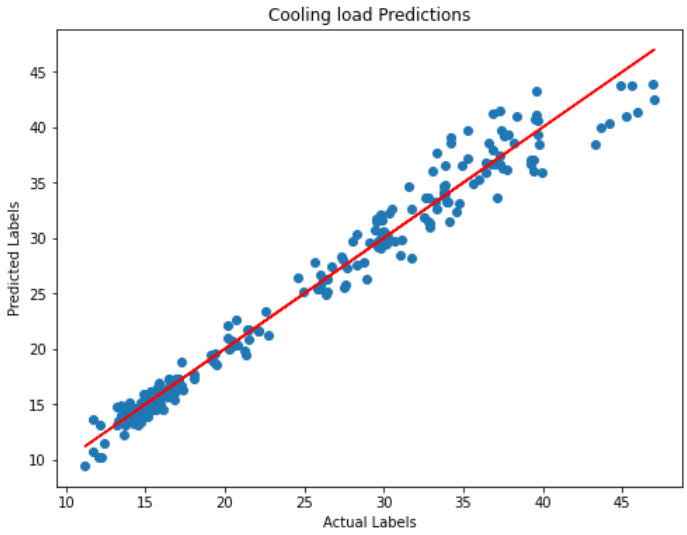
In [35]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score
print('Heating Load')
print('Evaluation MSE:\t {}'.format(mse(y_test1, y_pred1, squared='False')))
print('Explained score: {}'.format(explained_variance_score(y_test1, y_pred1)))

plt.figure(figsize=(8,6))
plt.scatter(y_test1, y_pred1)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Heating load Predictions')

plt.plot(y_test1, y_test1, color='red')
plt.show()
```

```
Heating Load
Evaluation MSE:  0.24245235038691273
Explained score: 0.9977711687193728
```



In [36]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score
print('Cooling Load')
print('Evaluation MSE:\t {}'.format(mse(y_test2, y_pred2, squared='False')))
print('Explained score: {}'.format(explained_variance_score(y_test2, y_pred2)))

plt.figure(figsize=(8,6))
plt.scatter(y_test2, y_pred2)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Cooling load Predictions')

plt.plot(y_test2, y_test2, color='red')
plt.show()
```

```
Cooling Load
Evaluation MSE:  2.5081070275579176
Explained score: 0.9736643895147131
```



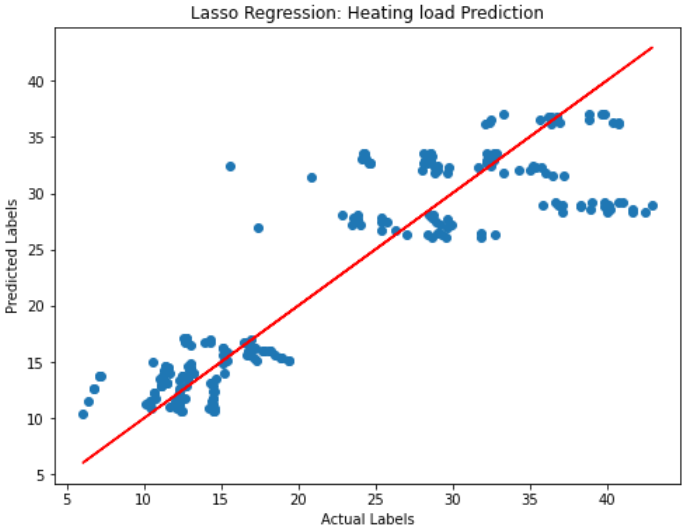
Case 3: Regulization Regression

a. LASSO Regression

In [37]:
```
from sklearn.linear_model import Lasso

# Train
lasso1 = Lasso()
lasso1.fit(X_train, y_train1)

lasso2 = Lasso()
lasso2.fit(X_train, y_train2)

# Predict
y_pred1 = lasso1.predict(X_test)
y_pred2 = lasso2.predict(X_test)
```

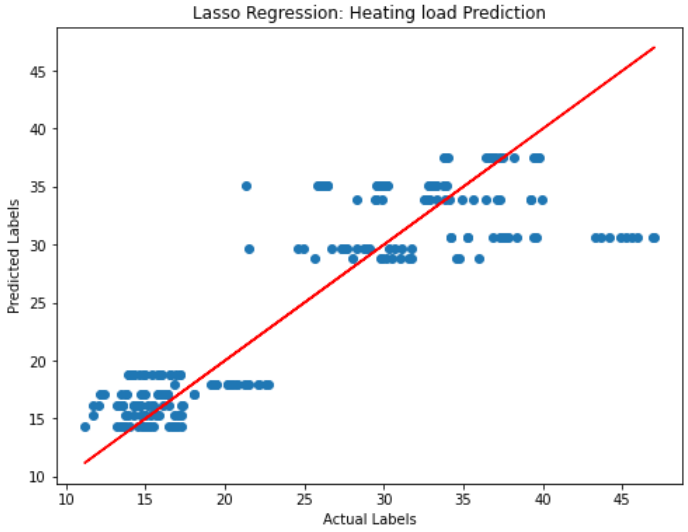
In [38]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score

print('Lasso Heating Load')
print('Lasso Test MSE: {}'.format(mse(y_test1, y_pred1, squared=5)))
print('Explained score: {}'.format(explained_variance_score(y_test1, y_pred1)))

plt.figure(figsize=(8,6))
plt.scatter(y_test1, y_pred1)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Lasso Regression: Heating load Prediction')

plt.plot(y_test1, y_test1, color='red')
plt.show()
```

```
Lasso Heating Load
Lasso Test MSE: 22.037265260760975
Explained score: 0.795019984290958
```



In [39]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score

print('Lasso Heating Load')
print('Lasso Test MSE: {}'.format(mse(y_test2, y_pred2, squared=5)))
print('Explained score: {}'.format(explained_variance_score(y_test2, y_pred2)))

plt.figure(figsize=(8,6))
plt.scatter(y_test2, y_pred2)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Lasso Regression: Heating load Prediction')

plt.plot(y_test2, y_test2, color='red')
plt.show()
```

```
Lasso Heating Load
Lasso Test MSE: 21.66489307653214
Explained score: 0.7716161573512936
```

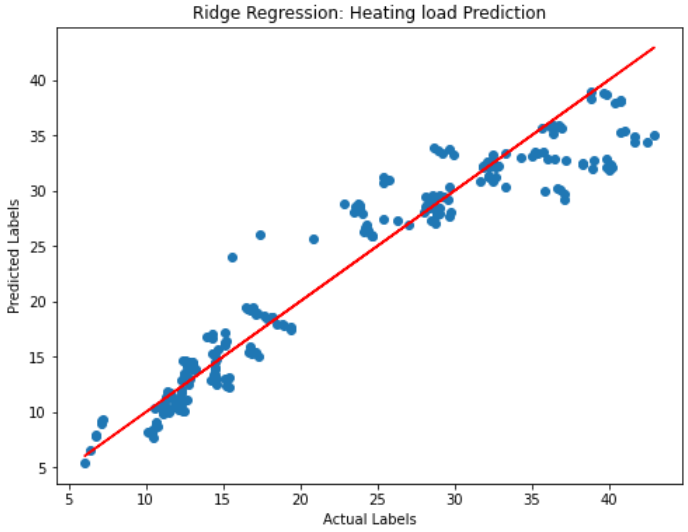


b. Ridge Regression

In [40]:
```
from sklearn.linear_model import Ridge

# Train
ridge1 = Ridge()
ridge1.fit(X_train, y_train1)

ridge2 = Ridge()
ridge2.fit(X_train, y_train2)

# Predict
y_pred1 = ridge1.predict(X_test)
y_pred2 = ridge2.predict(X_test)
```

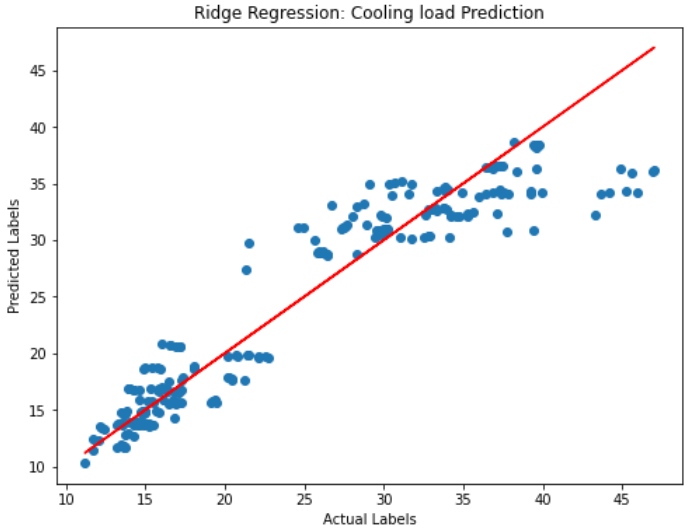
In [41]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score

print('Heating Load')
print('Ridge Test MSE: {}'.format(mse(y_test1, y_pred1, squared=False)))
print('Explained score: {}'.format(explained_variance_score(y_test1, y_pred1)))

plt.figure(figsize=(8,6))
plt.scatter(y_test1, y_pred1)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Ridge Regression: Heating load Prediction')

plt.plot(y_test1, y_test1, color='red')
plt.show()
```

```
Heating Load
Ridge Test MSE: 2.9146763715821744
Explained score: 0.9225404933129574
```



In [42]:
```
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score
import matplotlib.pyplot as plt

print('Cooling Load')
print('Ridge Test MSE: {}'.format(mse(y_test2, y_pred2, squared=False)))
print('Explained score: {}'.format(explained_variance_score(y_test2, y_pred2)))

plt.figure(figsize=(8,6))
plt.scatter(y_test2, y_pred2)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Ridge Regression: Cooling load Prediction')

plt.plot(y_test2, y_test2, color='red')
plt.show()
```

```
Cooling Load
Ridge Test MSE: 3.2164805690505323
Explained score: 0.8921572470259151
```

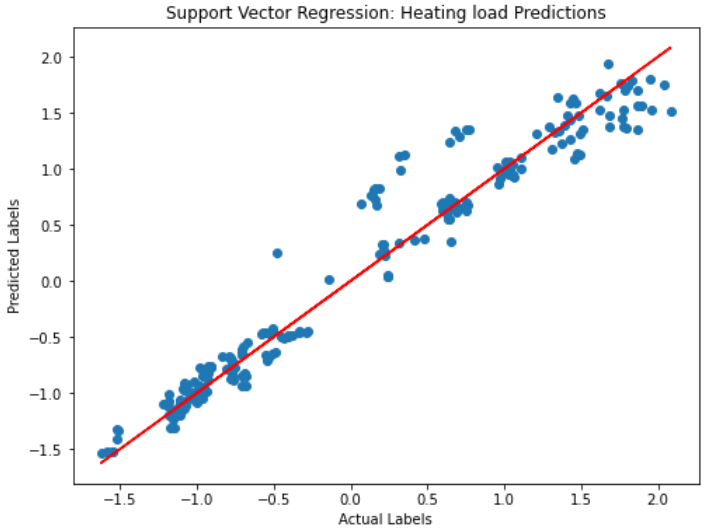


Case 4: Support Vector Regression (SVR)

In [43]:
```
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler


# Scale X and Y
X_scaler = StandardScaler()
X_train = X_scaler.fit_transform(X_train)
X_test = X_scaler.transform(X_test)

y1_scaler = StandardScaler()
y_train1 = y1_scaler.fit_transform(y_train1)
y_test1 = y1_scaler.transform(y_test1)

y2_scaler = StandardScaler()
y_train2 = y2_scaler.fit_transform(y_train2)
y_test2 = y2_scaler.transform(y_test2)

# Train
y_train1 = y_train1.reshape(-1,1)
y_train2 = y_train2.reshape(-1,1)

y_train1 = np.ravel(y_train1)
y_train2 = np.ravel(y_train2)


svr1 = SVR(kernel='rbf')
svr1.fit(X_train, y_train1)

svr2 = SVR(kernel='rbf')
svr2.fit(X_train, y_train2)

# Predict
y_pred1 = svr1.predict(X_test)
y_pred2 = svr2.predict(X_test)
```

In [44]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score

print('SVR Heating Load')
print('Evaluation MSE:\t {}'.format(mse(y_test1, y_pred1, squared='False')))
print('Explained score: {}'.format(explained_variance_score(y_test1, y_pred1)))

plt.figure(figsize=(8,6))
plt.scatter(y_test1, y_pred1)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Support Vector Regression: Heating load Predictions')

plt.plot(y_test1, y_test1, color='red')
plt.show()
```

```
SVR Heating Load
Evaluation MSE:  0.04928249856721011
Explained score: 0.9550994798153676
```



In [45]:
```
import matplotlib.pyplot as plt
from sklearn.metrics import mean_squared_error as mse
from sklearn.metrics import explained_variance_score

print('SVR Cooling Load')
print('Evaluation MSE:\t {}'.format(mse(y_test2, y_pred2, squared='False')))
print('Explained score: {}'.format(explained_variance_score(y_test2, y_pred2)))

plt.figure(figsize=(8,6))
plt.scatter(y_test2, y_pred2)
plt.xlabel('Actual Labels')
plt.ylabel('Predicted Labels')
plt.title('Support Vector Regression: Cooling load Predictions')

plt.plot(y_test2, y_test2, color='red')
plt.show()
```

```
SVR Cooling Load
Evaluation MSE:  0.08931678352443816
Explained score: 0.9169573680386633
```