In [16]:
```
#importing all required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.preprocessing import normalize
from imblearn.combine import SMOTEENN
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, mean_squared_error
from sklearn.metrics import precision_recall_curve
from sklearn.model_selection import cross_val_predict
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import RocCurveDisplay
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
import seaborn as sb
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import explained_variance_score
```

Classification: Credit Card Fraud

In [ ]:
```
df = pd.read_csv("creditcard.csv")

#removing infinite and NAN values
pd.set_option('mode.use_inf_as_na', True)
df.dropna(inplace = True)

df
```

Out[ ]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.27
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.63
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.77
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.00
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.79
...	...	...	...	...	...	...	...	...	...	...	...	...	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918215	7.305334	1.914428	...	0.213454	0.11
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024330	0.294869	0.584800	...	0.214205	0.92
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296827	0.708417	0.432454	...	0.232045	0.57
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686180	0.679145	0.392087	...	0.265245	0.80
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577006	-0.414650	0.486180	...	0.261057	0.64

284807 rows × 31 columns

In [ ]:
```
#since class represents whether a transaction is fradulent X will be Class and Y will be everything else
Y = df['Class']
X = df[df.columns[0:30]]

#using combination of undersampling and oversampling to better balance the data
smk = SMOTEENN(random_state = 42)
X_new, Y_new = smk.fit_resample(X,Y)

#splitting training and test set into 60-40 ratio
X_train, X_test, y_train, y_test = train_test_split(X_new,Y_new, test_size = 0.4, random_state = 42)
```

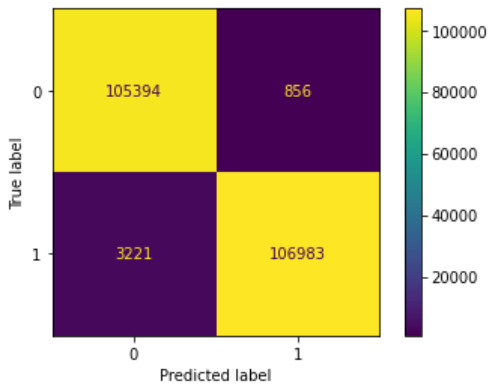
First Classification Technique: Logistic Regression

In [ ]:
```
#scaling dataset using a Standard Scaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

#use logistic regression
log_reg = LogisticRegression(solver = 'saga', random_state = 42, max_iter = 1000000)
log_reg.fit(X_train, y_train)
```

Out[ ]:
```
LogisticRegression(max_iter=1000000, random_state=42, solver='saga')
```

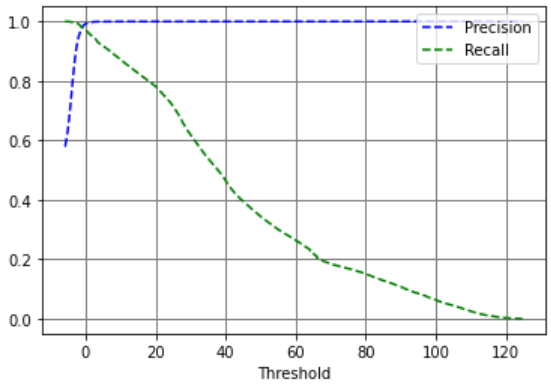
In [ ]:
```
#visualization

#1. confusion matrix

y_pred = log_reg.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm).plot()

print("accuracy score: ", accuracy_score(y_test, y_pred))
print("\nprecision score: ", precision_score(y_test, y_pred))
print ("\nrecall score: ", recall_score(y_test, y_pred))
print("\nf1 score: ", f1_score(y_test, y_pred))
```

```
accuracy score:  0.98116458924298

precision score:  0.992062240933243

recall score:  0.9707723857573228

f1 score:  0.9813018533041649
```



In [ ]:
```
#decision function
y_scores = cross_val_predict(log_reg, X_train, y_train, cv = 3, method = "decision_function")
precisions, recalls, thresholds = precision_recall_curve(y_train, y_scores)
```

In [ ]:
```
#visualising precision and recall as a function of thresholds
def plot_precision_recall_vs_thresholds(precisions, recalls, thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(loc='upper right')
    plt.grid(b=True, which="both", axis="both", color='gray', linestyle='-', linewidth=1)

plot_precision_recall_vs_thresholds(precisions, recalls, thresholds)
plt.show()
```

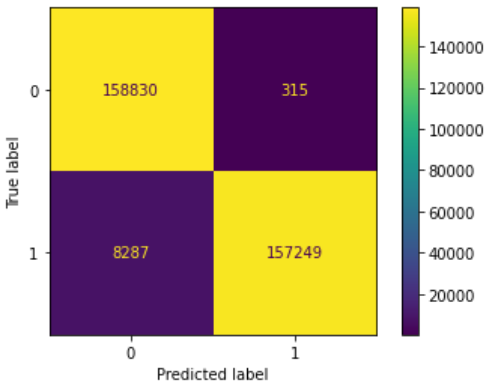


In [ ]:
```
threshold_recall = thresholds[np.argmax(precisions >= 0.998)]
print(threshold_recall)

y_train_pred = (y_scores >= threshold_recall)
```

```
1.9975108776904413
```

In [ ]:
```
#accuracy & precision metrics
print("accuracy score: ", accuracy_score(y_train, y_train_pred))
print("\nprecision score: ", precision_score(y_train, y_train_pred))
print ("\nrecall score: ", recall_score(y_train, y_train_pred))
print("\nf1 score: ", f1_score(y_train, y_train_pred))

#visualization with new confusion matrix
cm = confusion_matrix(y_train, y_train_pred)
print("\n\n")
disp = ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

```
accuracy score:  0.9735063031098217

precision score:  0.9980008123683075

recall score:  0.9499383819833752

f1 score:  0.9733766635716496
```



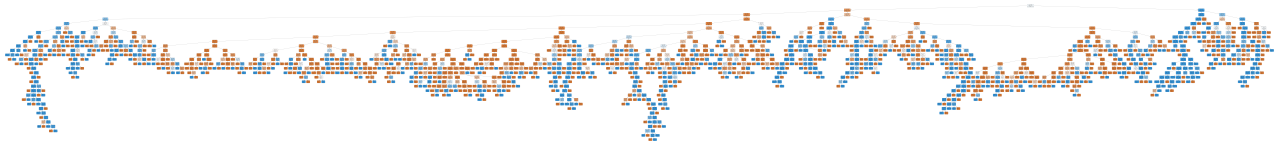
1. Random Forest Classifier

In [ ]:
```
rnd_clf = RandomForestClassifier(n_estimators = 500, random_state = 42)
rnd_clf.fit(X_train, y_train)
```

Out[ ]:
```
RandomForestClassifier(n_estimators=500, random_state=42)
```

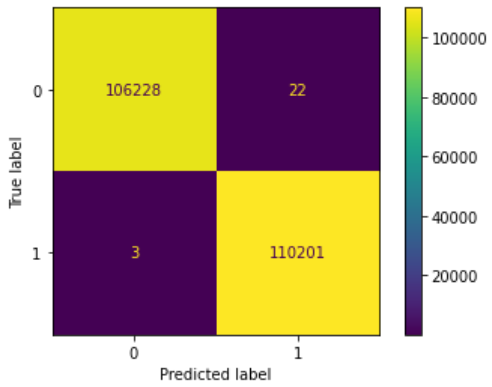
In [ ]:
```
#visualising one of our estimators
estimator = rnd_clf.estimators_[250]

from sklearn.tree import export_graphviz
# Export as dot file
export_graphviz(estimator, out_file='tree.dot',
                feature_names = df.columns[0:30],
                class_names = df.columns[30],
                rounded = True, proportion = False,
                precision = 3, filled = True)

# Convert to png using system command (requires Graphviz)
from subprocess import call
call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'tree.png')
```

Out[ ]:



In [ ]:
```
y_pred_rfc = rnd_clf.predict(X_test)
print("accuracy score: ", accuracy_score(y_test, y_pred_rfc))
print("\nprecision score: ", precision_score(y_test, y_pred_rfc))
print ("\nrecall score: ", recall_score(y_test, y_pred_rfc))
print("\nf1 score: ", f1_score(y_test, y_pred_rfc))

cm = confusion_matrix(y_test, y_pred_rfc)
disp = ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

```
accuracy score:  0.9998845020189047

precision score:  0.9998004046342415

recall score:  0.9999727777576132

f1 score:  0.9998865837669614
```



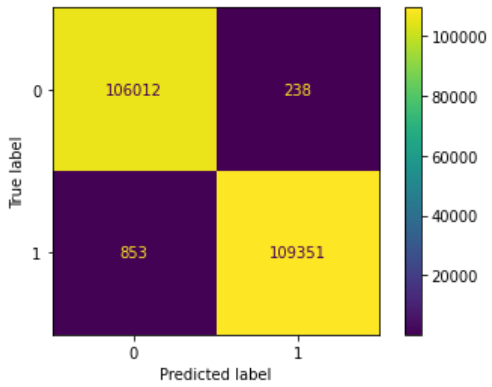
1. Support Vector Machine

In [ ]:
```
svm_clf = SVC(gamma = "scale", random_state= 42)
svm_clf.fit(X_train, y_train)
```

Out[ ]:
```
SVC(random_state=42)
```

In [ ]:
```
y_pred_svm = svm_clf.predict(X_test)
print("accuracy score: ", accuracy_score(y_test, y_pred_svm))
print("\nprecision score: ", precision_score(y_test, y_pred_svm))
print ("\nrecall score: ", recall_score(y_test, y_pred_svm))
print("\nf1 score: ", f1_score(y_test, y_pred_svm))

cm = confusion_matrix(y_test, y_pred_svm)
disp = ConfusionMatrixDisplay(confusion_matrix=cm).plot()
```

```
accuracy score:  0.9949596681050015

precision score:  0.9978282491855934

recall score:  0.99225980908134

f1 score:  0.9950362386427228
```



Regression: Energy Efficiency

In [3]:
```
df2 = pd.read_excel("ENB2012_data.xlsx")
df2
```

Out[3]:

	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

1. Linear Regression

In [12]:
```
#then splitting it into input and output variables
X2 = df2[['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8']]
Y2h = df2['Y1']
Y2c = df2['Y2']

#train-test-split
X_train_2h, X_test_2h, y_train_2h, y_test_2h = train_test_split(X2, Y2h, test_size=0.3)
X_train_2c, X_test_2c, y_train_2c, y_test_2c = train_test_split(X2, Y2c, test_size=0.3)
```

In [13]:
```
lin_reg_h = LinearRegression()
lin_reg_h.fit(X_test_2h, y_test_2h)
lin_reg_c = LinearRegression()
lin_reg_c.fit(X_test_2c, y_test_2c)

h_predict = lin_reg_h.predict(X_test_2h)
h_mse =  mean_squared_error(y_test_2h, h_predict, squared = True)
h_rmse = mean_squared_error(y_test_2h, h_predict, squared=False)
h_er = explained_variance_score(y_test_2h, h_predict);
print("For heating load:\n")
print("mse: ", h_mse, "\nrmse: ", h_rmse, "\n")
print("explained variance: ", h_er, "\n\n" )

c_predict = lin_reg_c.predict(X_test_2c)
c_mse =  mean_squared_error(y_test_2c, c_predict, squared = True)
c_rmse = mean_squared_error(y_test_2c, c_predict, squared=False)
c_er = explained_variance_score(y_test_2c, c_predict);
print("For cooling load:\n")
print("mse: ", c_mse, "\nrmse: ", c_rmse, "\n")
print("explained variance: ", c_er )
```

```
For heating load:

mse:  7.54175205482957
rmse:  2.746225055385951

explained variance:  0.9281357419727112


For cooling load:

mse:  9.07299916200867
rmse:  3.0121419558195908

explained variance:  0.8955523781001445
```

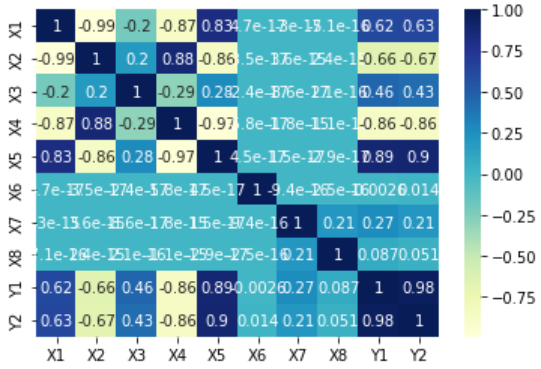
1. Ridge Regression

In [14]:
```
#performing the ridge regression

lin_ridge_h = Ridge(alpha= 0.001)
lin_ridge_h.fit(X_test_2h, y_test_2h)

lin_ridge_c = Ridge(alpha= 0.001)
lin_ridge_c.fit(X_test_2c, y_test_2c)

h_predict = lin_ridge_h.predict(X_test_2h)
h_mse =  mean_squared_error(y_test_2h, h_predict, squared = True)
h_rmse = mean_squared_error(y_test_2h, h_predict, squared=False)
h_er = explained_variance_score(y_test_2h, h_predict);
print("For heating load:\n")
print("mse: ", h_mse, "\nrmse: ", h_rmse, "\n")
print("explained variance: ", h_er, "\n\n" )

c_predict = lin_ridge_c.predict(X_test_2c)
c_mse =  mean_squared_error(y_test_2c, c_predict, squared = True)
c_rmse = mean_squared_error(y_test_2c, c_predict, squared=False)
c_er = explained_variance_score(y_test_2c, c_predict);
print("For cooling load:\n")
print("mse: ", c_mse, "\nrmse: ", c_rmse, "\n")
print("explained variance: ", c_er )
```

```
For heating load:

mse:  7.554846643021402
rmse:  2.7486081283117465

explained variance:  0.9280098614402169


For cooling load:

mse:  9.07376408707966
rmse:  3.0122689267526663

explained variance:  0.8955435723454905
```

regardless of alpha, mse and rmse are low enough that they can practically be considered 0, indicating that there is overfitting. to resolve this, I will reduce the number of features to the 2 with the highest correlation with heating load and cooling load, respectively

In [ ]:
```
dataplot = sb.heatmap(df2.corr(), cmap="YlGnBu", annot=True)
```



As evidenced from the heatmap, the features most correlated with Y1 (heat load) and Y2 (cooling load) are X4 and X5 because their absolute values are the greatest

In [ ]:
```
input = df2[['X1', 'X2', 'X3', 'X6', 'X7', 'X8']]
input_train_h, input_test_h, y2_train_h, y2_test_h = train_test_split(input, Y2h, test_size=0.3, random_state
= 42)
input_train_c, input_test_c, y2_train_c, y2_test_c = train_test_split(input, Y2c, test_size=0.3, random_state
= 42)

h_ridge = Ridge(alpha = .01)
c_ridge = Ridge(alpha = .01)
h_ridge.fit(input_test_h, y2_test_h)
c_ridge.fit(input_test_c, y2_test_c)

h_predict = h_ridge.predict(input_test_h)
h_mse =  mean_squared_error(y2_test_h, h_predict, squared = True)
h_rmse = mean_squared_error(y2_test_h, h_predict, squared=False)
print("For heating load:\n")
print("mse: ", h_mse, "\nrmse: ", h_rmse, "\n\n")

c_predict = c_ridge.predict(input_test_c)
c_mse =  mean_squared_error(y2_test_c, c_predict, squared = True)
c_rmse = mean_squared_error(y2_test_c, c_predict, squared=False)
print("For cooling load:\n")
print("mse: ", c_mse, "\nrmse: ", c_rmse)
```

```
For heating load:

mse:  10.653755071803959
rmse:  3.2640090489770337


For cooling load:

mse:  12.29989964404758
rmse:  3.507121275925254
```

1. Polynomial Regression

In [ ]:
```
#degree 2
poly_features = PolynomialFeatures(degree=2, include_bias = False)
X_poly_h = poly_features.fit_transform(X_test_2h)
X_poly_c = poly_features.fit_transform(X_test_2c)

h_ridge.fit(X_poly_h, y_test_2h)
c_ridge.fit(X_poly_c, y_test_2c)

predict_h = h_ridge.predict(X_poly_h)
square_mse_h = mean_squared_error(y_test_2h, predict_h, squared = True)
square_rmse_h = mean_squared_error(y_test_2h, predict_h, squared=False)

predict_c = c_ridge.predict(X_poly_c)
square_mse_c = mean_squared_error(y_test_2c, predict_c, squared = True)
square_rmse_c = mean_squared_error(y_test_2c, predict_c, squared=False)
```

In [ ]:
```
#degree 3
poly_features = PolynomialFeatures(degree=3, include_bias = False)
X_poly_h = poly_features.fit_transform(X_test_2h)
X_poly_c = poly_features.fit_transform(X_test_2c)

h_ridge.fit(X_poly_h, y_test_2h)
c_ridge.fit(X_poly_c, y_test_2c)

predict_h = h_ridge.predict(X_poly_h)
cube_mse_h = mean_squared_error(y_test_2h, predict_h, squared = True)
cube_rmse_h = mean_squared_error(y_test_2h, predict_h, squared=False)

predict_c = c_ridge.predict(X_poly_c)
cube_mse_c = mean_squared_error(y_test_2c, predict_c, squared = True)
cube_rmse_c = mean_squared_error(y_test_2c, predict_c, squared=False)
```

In [ ]:
```
#degree 4
poly_features = PolynomialFeatures(degree=4, include_bias = False)
X_poly_h = poly_features.fit_transform(X_test_2h)
X_poly_c = poly_features.fit_transform(X_test_2c)

h_ridge.fit(X_poly_h, y_test_2h)
c_ridge.fit(X_poly_c, y_test_2c)

predict_h = h_ridge.predict(X_poly_h)
degreefour_mse_h = mean_squared_error(y_test_2h, predict_h, squared = True)
degreefour_rmse_h = mean_squared_error(y_test_2h, predict_h, squared=False)

predict_c = c_ridge.predict(X_poly_c)
degreefour_mse_c = mean_squared_error(y_test_2c, predict_c, squared = True)
degreefour_rmse_c = mean_squared_error(y_test_2c, predict_c, squared=False)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:197: UserWarning: Singular matrix in so
lving dual problem. Using least-squares solution instead.
  "Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:197: UserWarning: Singular matrix in so
lving dual problem. Using least-squares solution instead.
  "Singular matrix in solving dual problem. Using "
```

In [ ]:
```
#degree 5
poly_features = PolynomialFeatures(degree=5, include_bias = False)
X_poly_h = poly_features.fit_transform(X_test_2h)
X_poly_c = poly_features.fit_transform(X_test_2c)

h_ridge.fit(X_poly_h, y_test_2h)
c_ridge.fit(X_poly_c, y_test_2c)

predict_h = h_ridge.predict(X_poly_h)
degreefive_mse_h = mean_squared_error(y_test_2h, predict_h, squared = True)
degreefive_rmse_h = mean_squared_error(y_test_2h, predict_h, squared=False)

predict_c = c_ridge.predict(X_poly_c)
degreefive_mse_c = mean_squared_error(y_test_2c, predict_c, squared = True)
degreefive_rmse_c = mean_squared_error(y_test_2c, predict_c, squared=False)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:197: UserWarning: Singular matrix in so
lving dual problem. Using least-squares solution instead.
  "Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:197: UserWarning: Singular matrix in so
lving dual problem. Using least-squares solution instead.
  "Singular matrix in solving dual problem. Using "
```

In [ ]:
```
#degree 8
poly_features = PolynomialFeatures(degree=8, include_bias = False)
X_poly_h = poly_features.fit_transform(X_test_2h)
X_poly_c = poly_features.fit_transform(X_test_2c)

h_ridge.fit(X_poly_h, y_test_2h)
c_ridge.fit(X_poly_c, y_test_2c)

predict_h = h_ridge.predict(X_poly_h)
degreeeight_mse_h = mean_squared_error(y_test_2h, predict_h, squared = True)
degreeeight_rmse_h = mean_squared_error(y_test_2h, predict_h, squared=False)

predict_c = c_ridge.predict(X_poly_c)
degreeeight_mse_c = mean_squared_error(y_test_2c, predict_c, squared = True)
degreeeight_rmse_c = mean_squared_error(y_test_2c, predict_c, squared=False)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:197: UserWarning: Singular matrix in so
lving dual problem. Using least-squares solution instead.
  "Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:197: UserWarning: Singular matrix in so
lving dual problem. Using least-squares solution instead.
  "Singular matrix in solving dual problem. Using "
```

In [ ]:

```
#degree 10
poly_features = PolynomialFeatures(degree=2, include_bias = False)
X_poly_h = poly_features.fit_transform(X_test_2h)
X_poly_c = poly_features.fit_transform(X_test_2c)

h_ridge.fit(X_poly_h, y_test_2h)
c_ridge.fit(X_poly_c, y_test_2c)

predict_h = h_ridge.predict(X_poly_h)
degreeten_mse_h = mean_squared_error(y_test_2h, predict_h, squared = True)
degreeten_rmse_h = mean_squared_error(y_test_2h, predict_h, squared=False)

predict_c = c_ridge.predict(X_poly_c)
degreeten_mse_c = mean_squared_error(y_test_2c, predict_c, squared = True)
degreeten_rmse_c = mean_squared_error(y_test_2c, predict_c, squared=False)
```

In [ ]:

```
#comparing metrics
print("Degree = 2:\n")
print("Heating: \n")
print("mse: ", square_mse_h, "\nrmse: ", square_rmse_h, "\n")
print("Cooling: \n")
print("mse: ", square_mse_c, "\nrmse: ", square_rmse_c, "\n")

print("----------------------------------------------------------------------")

print("Degree = 3:\n")
print("Heating: \n")
print("mse: ", cube_mse_h, "\nrmse: ", cube_rmse_h, "\n")
print("Cooling: \n")
print("mse: ", cube_mse_c, "\nrmse: ", cube_rmse_c)

print("----------------------------------------------------------------------")

print("\nDegree = 4:\n")
print("Heating: \n")
print("mse: ", degreefour_mse_h, "\nrmse: ", degreefour_rmse_h, "\n")
print("Cooling: \n")
print("mse: ", degreefour_mse_c, "\nrmse: ", degreefour_rmse_c)

print("----------------------------------------------------------------------")

print("\nDegree = 5:\n")
print("Heating: \n")
print("mse: ", degreefive_mse_h, "\nrmse: ", degreefive_rmse_h, "\n")
print("Cooling: \n")
print("mse: ", degreefive_mse_c, "\nrmse: ", degreefive_rmse_c)

print("----------------------------------------------------------------------")

print("\nDegree = 8:\n")
print("Heating: \n")
print("mse: ", degreeeight_mse_h, "\nrmse: ", degreeeight_rmse_h, "\n")
print("Cooling: \n")
print("mse: ", degreeeight_mse_c, "\nrmse: ", degreeeight_rmse_c)

print("----------------------------------------------------------------------")

print("\nDegree = 10:\n")
print("Heating: \n")
print("mse: ", degreeten_mse_h, "\nrmse: ", degreeten_rmse_h, "\n")
print("Cooling: \n")
print("mse: ", degreeten_mse_c, "\nrmse: ", degreeten_rmse_c, "\n")
```

```
Degree = 2:

Heating:

mse:  2.234470896708472
rmse:  1.4948146696860023

Cooling:

mse:  3.9955291174592404
rmse:  1.9988819668652875

----------------------------------------------------------------------
Degree = 3:

Heating:

mse:  0.1858926428557722
rmse:  0.4311526908831397

Cooling:

mse:  1.1671215767898735
rmse:  1.080334011678737
----------------------------------------------------------------------

Degree = 4:

Heating:

mse:  0.5328945809838199
rmse:  0.7299962883356462

Cooling:

mse:  1.6216187677147174
rmse:  1.2734279593737203
----------------------------------------------------------------------

Degree = 5:

Heating:

mse:  0.533673927297389
rmse:  0.7305298948690526

Cooling:

mse:  1.772104700765512
rmse:  1.3312042295476347
----------------------------------------------------------------------

Degree = 8:

Heating:

mse:  2.7295682540856747
rmse:  1.6521405067625679

Cooling:

mse:  2.384349265598429
rmse:  1.5441338237336908
----------------------------------------------------------------------

Degree = 10:

Heating:

mse:  2.234470896708472
rmse:  1.4948146696860023

Cooling:

mse:  3.9955291174592404
```

```
rmse:  1.9988819668652875
```

In [17]:
```
%%shell
jupyter nbconvert --to html
```

```
[NbConvertApp] WARNING | pattern '/content/Lynch_Final_Project.ipynb' matched no files
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and include the error message in the
cell output (the default behaviour is to abort conversion). This flag is only relevant if '--execute' was spe
cified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWrite
r.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False --NbConvertApp.export_format=notebook --FilesWrite
r.build_directory= --ClearOutputPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True --TemplateExporter.exclude_output_prompt=Tru
e]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True --TemplateExporter.exclude_input=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR', 'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python', 'rst', 'script',
'slides']
            or a dotted object name that represents the import path for an
            `Exporter` class
    Default: 'html'
```

```
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template file to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_file]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    overwrite base name use for output files.
                can only be used when converting one notebook at a time.
    Default: ''
    Equivalent to: [--NbConvertApp.output_base]
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                    to output to the directory of each notebook. To recover
                                    previous default behaviour (outputting to the current
                                    working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb

            which will convert mynotebook.ipynb to the default format (probably HTML).

            You can specify the export format with `--to`.
            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook', 'pdf', 'python',
'rst', 'script', 'slides'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX includes
            'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
            can specify the flavor of the format used.

            > jupyter nbconvert --to html --template basic mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf
```

```
            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.


---------------------------------------------------------------------------
CalledProcessError                        Traceback (most recent call last)
<ipython-input-17-3fa74a6155b8> in <module>()
----> 1 get_ipython().run_cell_magic('shell', '', 'jupyter nbconvert --to html /content/Lynch_Final_Project.ipynb')

/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py in run_cell_magic(self, magic_name, line, cell)
   2115             magic_arg_s = self.var_expand(line, stack_depth)
   2116             with self.builtin_trap:
-> 2117                 result = fn(magic_arg_s, cell)
   2118             return result
   2119

/usr/local/lib/python3.7/dist-packages/google/colab/_system_commands.py in _shell_cell_magic(args, cmd)
    111   result = _run_command(cmd, clear_streamed_output=False)
    112   if not parsed_args.ignore_errors:
--> 113     result.check_returncode()
    114   return result
    115

/usr/local/lib/python3.7/dist-packages/google/colab/_system_commands.py in check_returncode(self)
    137     if self.returncode:
    138       raise subprocess.CalledProcessError(
--> 139           returncode=self.returncode, cmd=self.args, output=self.output)
    140
    141   def _repr_pretty_(self, p, cycle):  # pylint:disable=unused-argument

CalledProcessError: Command 'jupyter nbconvert --to html /content/Lynch_Final_Project.ipynb' returned non-zero exit status 255.
```