In [ ]:

```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
```

In [ ]:

```
#combination function(required for finding coefficients of Legendre's polynomial)
def comb(n, k):
  return math.factorial(n)/(math.factorial(k)*math.factorial(n-k))
```

In [ ]:

```
#Generating coefficients of required Legendre's polynomial and normalizing it
def generate_coeff(order):
  coeff = np.zeros((order+1,1)) #coefficients
  for i in range(int(order/2)+1):
    coeff[order - 2*i] = (-1)**i*comb(order, i)*comb(2*order - 2*i, order)/2**order
    coeff = coeff/np.linalg.norm(coeff)
    #coeff[2] = coeff[2] + 2
  return coeff
```

In [ ]:

```
#function generating data points given sample size, coefficients of polynomial, order of po
def generate_data(size, coeff, order, add_err = False):
  X = np.random.uniform(-7, 7, size)                                       #uniformly
  X = np.array( sorted( X ))
  #X = X/max(X)
  X = np.array([X**i for i in range(0,order+1)]).T                         #generate n

  coeff = coeff.reshape(-1,1)

  y = X @ coeff                                                            #generating
  if(add_err == True): y = y + np.random.normal(0,0.1, size).reshape(-1,1)      #adding noi

  return X[:,1], y
```

In [ ]:

```
#Function to compute lms loss
def error(a, b):
  return (1/2)*(np.linalg.norm(a-b))**2
```

In [ ]:

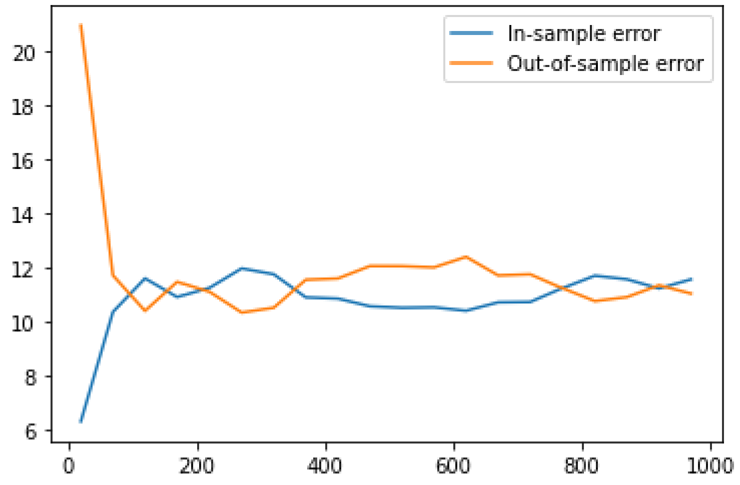
```
#defining function for fitting polynomial of given order given train and test data
def poly_fit(data, order, y, dtest, ytest):
  poly = PolynomialFeatures(degree = order)
  X_poly = poly.fit_transform(data)
  poly.fit(X_poly, y)
  lin2 = LinearRegression()                                               #using skle
  lin2.fit(X_poly, y)
  y_pred = lin2.predict(poly.fit_transform(dtest))
  Ein = error(y, lin2.predict(poly.fit_transform(data)))
  Eout = error(y_pred, ytest)                                             #calculatin
  return Ein, Eout
```

In [ ]:

```
#defining caltech1 function which will generate data as per need and will calculate expecte
def caltech1(poly_order, err, curve_order):
  c = generate_coeff(poly_order)                                          #generating
  E1 = []
  E2 = []
  for n in range(20, 1000, 50):
    ein = []
    eout = []
    for j in range(1000):
      x, y = generate_data(n, c, poly_order, add_err = err)               #generating
      x,y = x.reshape(-1,1), y.reshape(-1,1)
      X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.50, random_stat
      Ein, Eout = poly_fit(X_train, curve_order, y_train, X_test, y_test)      #fitting
      ein.append(Ein)
      eout.append(Eout)
    a = np.mean(ein)/n                                                    #mean error
    b = np.mean(eout)/n
    E1.append(a)
    E2.append(b)
  return E1, E2
    #print(n)
```

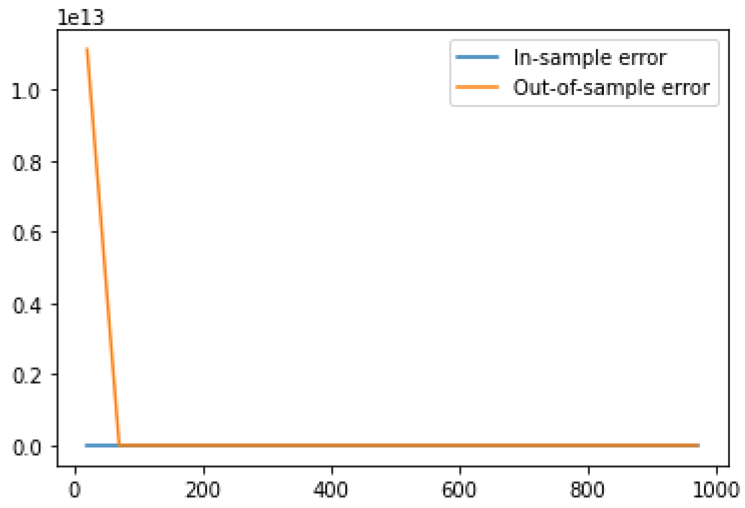
```
In [ ]:
```

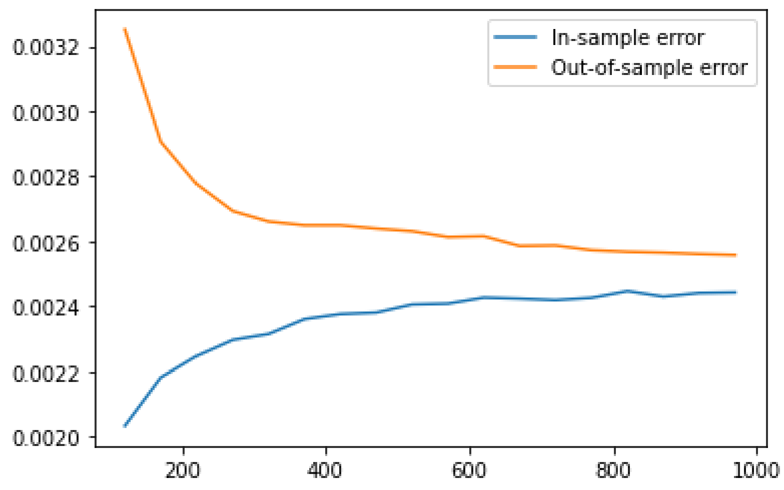
```
#plots for order 10 noisy data and quadratic fitting
E1, E2 = caltech1(10, True, 2)
plt.plot(range(20, 1000, 50),E1, label= 'In-sample error')
plt.plot(range(20, 1000, 50),E2, label= 'Out-of-sample error')
plt.legend()
plt.show()
```



In [ ]:

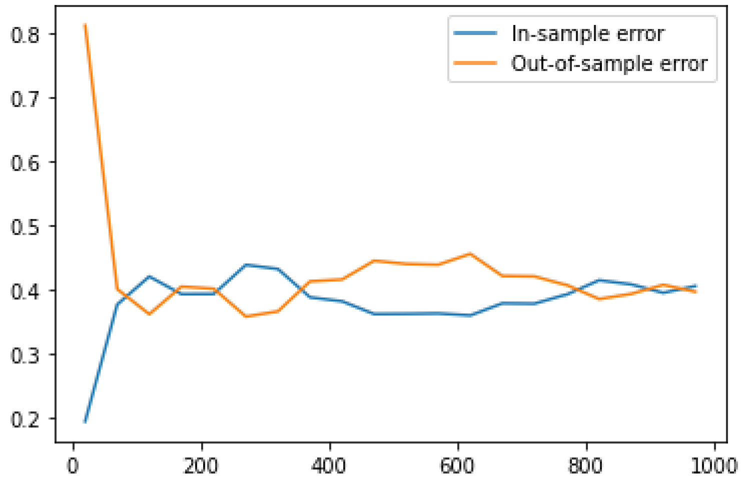
```
#plots for order 10 noisy data and order 10 polynomial fitting
E1, E2 = caltech1(10, True, 10)
plt.plot(range(20, 1000, 50),E1, label= 'In-sample error')
plt.plot(range(20, 1000, 50),E2, label= 'Out-of-sample error')
plt.legend()
plt.show()

plt.plot(range(120, 1000, 50),E1[2:], label= 'In-sample error')
plt.plot(range(120, 1000, 50),E2[2:], label= 'Out-of-sample error')
plt.legend()
plt.show()
```





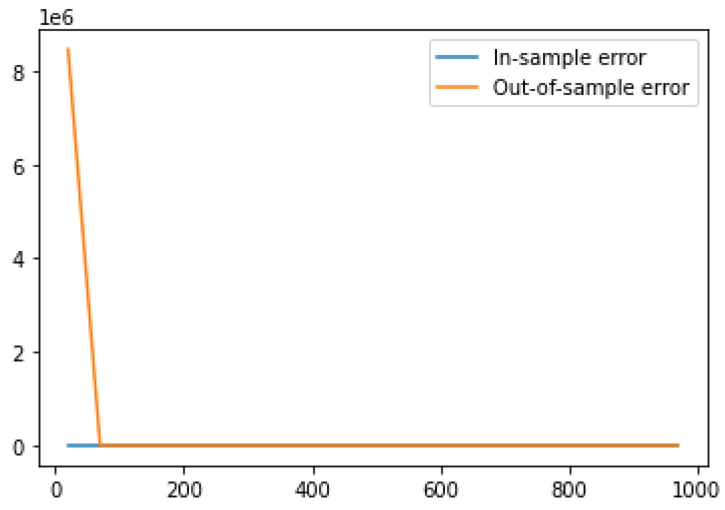
In [ ]:

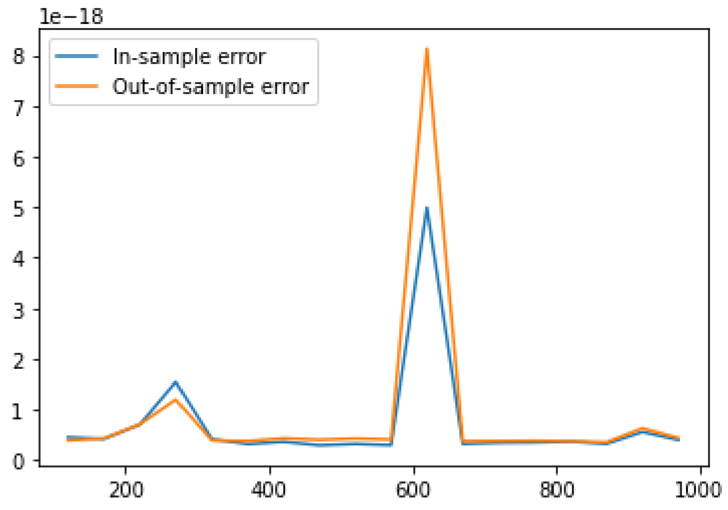
```
#plots for order 50 noisless data and quadratic polynomial fitting
E1, E2 = caltech1(50, False, 2)
plt.plot(range(20, 1000, 50),E1, label= 'In-sample error')
plt.plot(range(20, 1000, 50),E2, label= 'Out-of-sample error')
plt.legend()
plt.show()
```



In [ ]:

```python
#plots for order 50 noisless data and order 10 polynomial fitting
E1, E2 = caltech1(50, False, 10)
plt.plot(range(20, 1000, 50),E1, label= 'In-sample error')
plt.plot(range(20, 1000, 50),E2, label= 'Out-of-sample error')
plt.legend()
plt.show()

plt.plot(range(120, 1000, 50),E1[2:], label= 'In-sample error')
plt.plot(range(120, 1000, 50),E2[2:], label= 'Out-of-sample error')
plt.legend()
plt.show()
```





Observations and Conclusions:

1. In the first case of noisy data, initially, the test error(out of sample error) was lower for quadratic model even though train error(in sample error) was lower for 10th degree polynomial model indicating 10 degree model was 'overfitting' initially. The test error was in the range of 10^16 for 10th degree model.
2. As we go on increasing data points, for both models in-sample error started increasing and out of sample error started decreasing.
3. Both started converging to some fixed value which is the mean error that will remain in model.
4. This mean in higher for degree 2 as we are fitting degree 2 model for degree 10 data plus some noise is there whereas in case of degree 10 model mean error is only because of noise so quite less.
5. For second case of noiseless 50th order target, again similar observations followed just here mean error is because of higher order target.

6. So, conclusion is only for sufficiently large data, more complex model would give better results else it may 'overfit' to data. Also, noise is not only cause of 'overfitting'.