```
In [1]: import pandas as pd
        import numpy as np
        import pandas as pd
        from pandas import DataFrame
        import matplotlib.pyplot as plt
        import math
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import Ridge
        from sklearn import linear_model
        from sklearn.linear_model import LinearRegression
        from sklearn.preprocessing import PolynomialFeatures
        from sklearn.metrics import mean_squared_error
        from sklearn.pipeline import make_pipeline
        from sklearn.pipeline import Pipeline
        %matplotlib inline
```

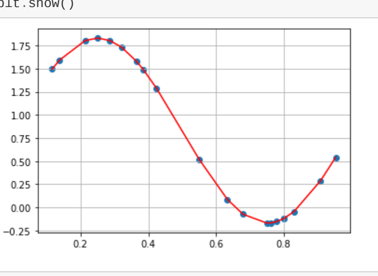
Generate 20 data pairs (X, Y) using y = sin(2*pi*X) + N

Use uniform distribution between 0 and 1 for X

Sample N from the normal gaussian distribution

Use 10 for train and 10 for test

```
In [2]: pi = np.pi
        x = np.random.uniform(0,1,20)
        n = np.random.normal(0,1)

In [3]: x.sort()
        x

Out[3]: array([0.11550161, 0.1381619 , 0.21395482, 0.25090457, 0.28618993,
               0.32115972, 0.36554996, 0.38517053, 0.4239901 , 0.55018   ,
               0.63356503, 0.67991725, 0.75030227, 0.7622195 , 0.77798733,
               0.80015235, 0.80105753, 0.83039906, 0.90896808, 0.95325508])

In [4]: n

Out[4]: 0.8306261964867722

In [5]: y = (np.sin(2*pi*x) + n)

In [6]: y

Out[6]: array([ 1.49429892,  1.59372652,  1.80508937,  1.83061004,  1.80488474,
                1.73232672,  1.57844764,  1.49113396,  1.29026127,  0.52053376,
                0.08650805, -0.07397935, -0.169372  , -0.16642787, -0.15395206,
               -0.12013407, -0.11835602, -0.04446979,  0.28933629,  0.54112371])

In [7]: plt.scatter(x, y)
        plt.plot(x,y, color = 'r')
        plt.grid(True)
        plt.show()
```



```
In [8]: X_train, X_test, y_train, y_test = train_test_split(x,y,test_size = 0.5)
```

Conversion of nd array to lists in Python

```
In [9]: X_train_list = X_train.tolist()
        X_test_list = X_test.tolist()
        y_train_list = y_train.tolist()
        y_test_list = y_test.tolist()
```

Conversion of lists to dictionary

```
In [10]: dictionary1 = {}
         for key in X_train_list:
             for value in y_train_list:
                 dictionary1[key] = value
                 y_train_list.remove(value)
                 break

         print(dictionary1)

{0.8010575331717582: -0.1183560215193814, 0.21395482374900998: 1.8050893705896334, 0.11550161
298060535: 1.4942989189098674, 0.5501800040124631: 0.5205337564254262, 0.3211597162653854: 1.
7323267239288151, 0.8303990635426058: -0.04446978571652804, 0.6335650292943724: 0.08650804608
684282, 0.2861899290796672: 1.804884739755416, 0.8001523531275022: -0.12013407358802652, 0.67
99172474627037: -0.07397934993932376}

In [11]: dictionary2 = {}
         for key in X_test_list:
             for value in y_test_list:
                 dictionary2[key] = value
                 y_test_list.remove(value)
                 break

         print(dictionary2)

{0.4239901048227982: 1.2902612745334545, 0.7622195039617758: -0.1664278659046391, 0.138161897
26897354: 1.593726517999295, 0.777987334568841: -0.15395206312880128, 0.385170530341859: 1.49
11339580407392, 0.3655499575812632: 1.5784476388723234, 0.9532550757862365: 0.541123714119315
9, 0.25090457298704916: 1.830610044877468, 0.7503022747653951: -0.1693719999415947, 0.9089680
789551647: 0.2893362904883191}
```

Creating sorted lists

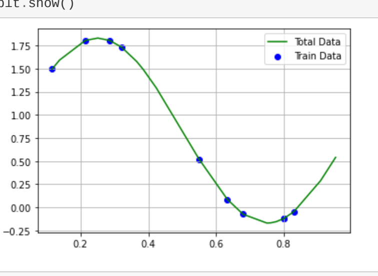
```
In [12]: sort_orders1 = sorted(dictionary1.items())
         sort_orders1

Out[12]: [(0.11550161298060535, 1.4942989189098674),
          (0.21395482374900998, 1.8050893705896334),
          (0.2861899290796672, 1.804884739755416),
          (0.3211597162653854, 1.7323267239288151),
          (0.5501800040124631, 0.5205337564254262),
          (0.6335650292943724, 0.08650804608684282),
          (0.6799172474627037, -0.07397934993932376),
          (0.8001523531275022, -0.12013407358802652),
          (0.8010575331717582, -0.1183560215193814),
          (0.8303990635426058, -0.04446978571652804)]

In [13]: sort_orders2 = sorted(dictionary2.items())
         sort_orders2

Out[13]: [(0.13816189726897354, 1.593726517999295),
          (0.25090457298704916, 1.830610044877468),
          (0.3655499575812632, 1.5784476388723234),
          (0.385170530341859, 1.4911339580407392),
          (0.4239901048227982, 1.2902612745334545),
          (0.7503022747653951, -0.1693719999415947),
          (0.7622195039617758, -0.1664278659046391),
          (0.777987334568841, -0.15395206312880128),
          (0.9089680789551647, 0.2893362904883191),
          (0.9532550757862365, 0.5411237141193159)]

In [14]: sorted_x = []
         sorted_y = []
         for i in sort_orders1:
             sorted_x.append(i[0])
             sorted_y.append(i[1])

In [15]: sorted_x2 = []
         sorted_y2 = []
         for i in sort_orders2:
             sorted_x2.append(i[0])
             sorted_y2.append(i[1])
```

Conversion of lists to Arrays

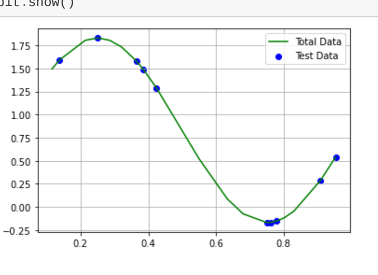
```
In [16]: X_train_sorted = np.array(sorted_x)
         Y_train_sorted = np.array(sorted_y)
         X_test_sorted = np.array(sorted_x2)
         Y_test_sorted = np.array(sorted_y2)
```

Created a Graph with X_tain and Y_Train

```
In [17]: plt.scatter(X_train_sorted, Y_train_sorted, color = 'b',label = 'Train Data')
         plt.plot(x,y, color= 'g', label = 'Total Data')
         plt.legend()
         plt.grid(True)
         plt.show()
```




```
In [18]: plt.scatter(X_test_sorted, Y_test_sorted, color = 'b',label = 'Test Data')
         plt.plot(x,y, color= 'g', label = 'Total Data')
         plt.legend()
         plt.grid(True)
         plt.show()
```



Using root mean square error, find weights of polynomial regression for order is 0, 1, 3, 9

Creating weights

```
In [19]: w=dict()
         degrees = [0,1,3,9]
         for degree in (degrees):
             w[degree] = (np.polyfit(X_train_sorted,Y_train_sorted,degree)).tolist()

         print(w)

{0: [0.7086702324932741], 1: [-3.1230453392140336, 2.342671699072447], 3: [29.16105928759116,
-43.2696436909863, 15.515318108258619, 0.22045191090303498], 9: [-37.03202601571583, 166.6015
071525459, -257.0559746632773, 122.32987090921816, 26.404095072058492, 16.80396488005687, -4
4.740355510592494, 0.4353243075647953, 6.251464974493874, 0.8316174160185423]}
```

Display weights in table

Display weights in tabular format

```
In [20]: df = pd.DataFrame.from_dict(w, orient='index').T
         df = df.fillna(value= 0)
         df = df.rename(columns ={0: 'M=0', 1 :'M=1', 3: 'M=3', 9:'M=9'})
         df = df.rename({0 :'w0', 1:'w1', 2:'w2', 3:'w3', 4:'w4', 5:'w5', 6:'w6', 7:'w7', 8:'w8', 9:
         'w9'})
         df
```

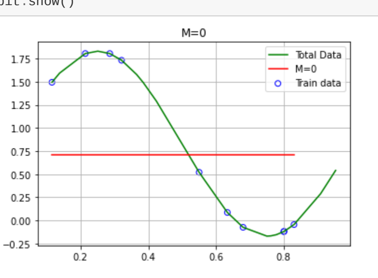
Out[20]:

	M=0	M=1	M=3	M=9
w0	0.70867	-3.123045	29.161059	-37.032026
w1	0.00000	2.342672	-43.269644	166.601507
w2	0.00000	0.000000	15.515318	-257.055975
w3	0.00000	0.000000	0.220452	122.329871
w4	0.00000	0.000000	0.000000	26.404095
w5	0.00000	0.000000	0.000000	16.803965
w6	0.00000	0.000000	0.000000	-44.740356
w7	0.00000	0.000000	0.000000	0.435324
w8	0.00000	0.000000	0.000000	6.251465
w9	0.00000	0.000000	0.000000	0.831617

Draw a chart of fit data

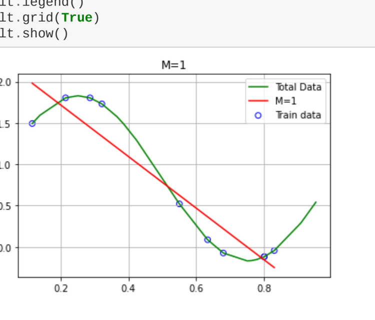
For M =0

```
In [21]: lin_reg = linear_model.LinearRegression()

In [22]: poly0=PolynomialFeatures(degree=0)
         x_poly0 = poly0.fit_transform(X_train_sorted.reshape(-1,1))
         poly0.fit(X_train_sorted.reshape(-1,1),Y_train_sorted)
         lin_reg.fit(x_poly0,Y_train_sorted)

Out[22]: LinearRegression()

In [23]: plt.scatter(X_train,y_train,color='blue',facecolors = 'none',edgecolors = 'b', label = 'Trai
         n data')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(X_train_sorted,lin_reg.predict(poly0.fit_transform(X_train_sorted.reshape(-1,1))),c
         olor='red', label = 'M=0')
         plt.title('M=0')
         plt.legend()
         plt.grid(True)
         plt.show()
```



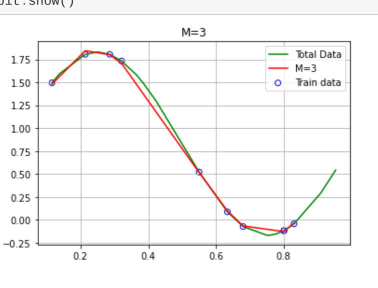
For M = 1

```
In [24]: poly1=PolynomialFeatures(degree=1)
         x_poly1 = poly1.fit_transform(X_train_sorted.reshape(-1,1))
         poly1.fit(X_train_sorted.reshape(-1,1),Y_train_sorted)
         lin_reg.fit(x_poly1,Y_train_sorted)

Out[24]: LinearRegression()

In [25]: plt.scatter(X_train,y_train,color='blue',facecolors = 'none',edgecolors = 'b', label = 'Trai
         n data')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(X_train_sorted,lin_reg.predict(poly1.fit_transform(X_train_sorted.reshape(-1,1))),c
         olor='red', label = 'M=1')
         plt.title('M=1')
         plt.legend()
         plt.grid(True)
         plt.show()
```



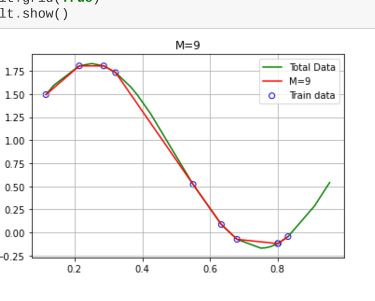
For M = 3

```
In [26]: poly3=PolynomialFeatures(degree=3)
         x_poly3 = poly3.fit_transform(X_train_sorted.reshape(-1,1))
         poly3.fit(X_train_sorted.reshape(-1,1),Y_train_sorted)
         lin_reg.fit(x_poly3,Y_train_sorted)

Out[26]: LinearRegression()

In [27]: plt.scatter(X_train,y_train,color='blue',facecolors = 'none',edgecolors = 'b', label = 'Trai
         n data')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(X_train_sorted,lin_reg.predict(poly3.fit_transform(X_train_sorted.reshape(-1,1))),c
         olor='red', label = 'M=3')
         plt.title('M=3')
         plt.legend()
         plt.grid(True)
         plt.show()
```



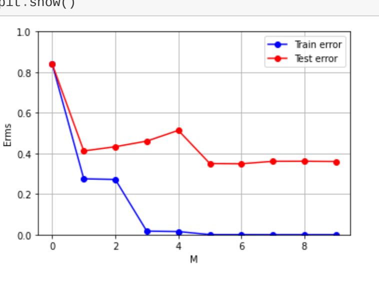
For M = 9

```
In [28]: poly9=PolynomialFeatures(degree=9)
         x_poly9 = poly9.fit_transform(X_train_sorted.reshape(-1,1))
         poly9.fit(X_train_sorted.reshape(-1,1),Y_train_sorted)
         lin_reg.fit(x_poly9,Y_train_sorted)

Out[28]: LinearRegression()

In [29]: plt.scatter(X_train,y_train,color='blue',facecolors = 'none',edgecolors = 'b', label = 'Trai
         n data')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(X_train_sorted,lin_reg.predict(poly9.fit_transform(X_train_sorted.reshape(-1,1))),c
         olor='red', label = 'M=9')
         plt.title('M=9')
         plt.legend()
         plt.grid(True)
         plt.show()
```



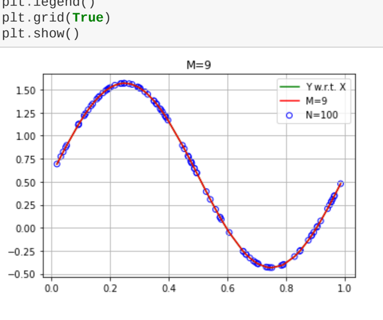
Draw train error vs test error

```
In [30]: train_error = np.empty(10)
         test_error = np.empty(10)

In [31]: for degree in range(10):
             est = make_pipeline(PolynomialFeatures(degree), LinearRegression())
             est.fit(X_train_sorted.reshape(-1,1), Y_train_sorted)
             train_error[degree] = np.sqrt(mean_squared_error(Y_train_sorted, est.predict(X_train_sor
         ted.reshape(-1,1))))
             test_error[degree] = np.sqrt(mean_squared_error(Y_train_sorted, est.predict(X_test_sorte
         d.reshape(-1,1))))

In [32]: train_error

Out[32]: array([8.39168433e-01, 2.74784593e-01, 2.71694031e-01, 1.70978257e-02,
                1.50662103e-02, 2.38984411e-04, 2.29822680e-04, 1.45194482e-06,
                1.11690026e-07, 7.56778050e-15])

In [33]: test_error

Out[33]: array([0.83916843, 0.41215655, 0.43283411, 0.46054198, 0.51347188,
                0.35017743, 0.34903408, 0.36104048, 0.36126409, 0.36006323])

In [34]: plt.plot(np.arange(10), train_error, 'bo-',color='blue', label='Train error')
         plt.plot(np.arange(10), test_error,'ro-', color='red', label='Test error')
         plt.ylim((0, 1))
         plt.ylabel('Erms')
         plt.xlabel('M')
         plt.legend()
         plt.grid(True)
         plt.show()
```



-> Now generate 100 more data and fit 9th order model and draw fit

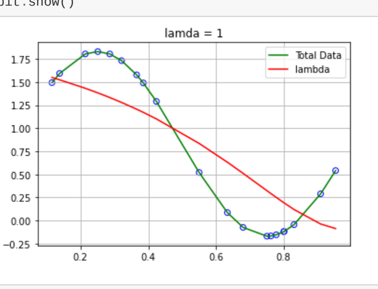
```
In [35]: pi2 = np.pi
         x2 = np.random.uniform(0,1,100)
         n2 = np.random.normal(0,1)

In [36]: x2.sort()
         x2

Out[36]: array([0.01969397, 0.0327831 , 0.0411304 , 0.04936523, 0.05294078,
                0.09235934, 0.09367728, 0.09423673, 0.1121785 , 0.11370472,
                0.11713743, 0.12610744, 0.13626578, 0.14148275, 0.15543447,
                0.16083153, 0.16454213, 0.1766924 , 0.18569411, 0.1904522 ,
                0.19481671, 0.19658359, 0.20129396, 0.21707923, 0.23661166,
                0.24030386, 0.24824081, 0.24912408, 0.26561488, 0.27558019,
                0.294847  , 0.29701924, 0.30132373, 0.30320114, 0.31883866,
                0.32885738, 0.34945382, 0.35052454, 0.35730205, 0.35962331,
                0.37143496, 0.37424227, 0.37854227, 0.38710301, 0.39007901,
                0.39205038, 0.39764239, 0.44706318, 0.45337731, 0.46606144,
                0.46795808, 0.47553108, 0.47707277, 0.47878355, 0.48747728,
                0.48813388, 0.49556089, 0.49565313, 0.52803102, 0.54191758,
                0.56008443, 0.57506593, 0.57529985, 0.57921643, 0.60628158,
                0.65351174, 0.65458086, 0.6625278 , 0.66468776, 0.67664863,
                0.68941975, 0.69591107, 0.70185503, 0.70344065, 0.70509181,
                0.7323155 , 0.73777416, 0.74520589, 0.75192166, 0.78433533,
                0.78904813, 0.79220434, 0.82874514, 0.84626971, 0.84891176,
                0.87573973, 0.88620859, 0.88797997, 0.89418198, 0.90495157,
                0.90683184, 0.91783125, 0.94121042, 0.94778671, 0.95360244,
                0.95369152, 0.95827882, 0.96339847, 0.96492496, 0.98568487])

In [37]: n2

Out[37]: 0.5680619793039537

In [38]: y2 = (np.sin(2*pi*x2) + n2)
         y2

Out[38]: array([ 0.69148733,  0.77259077,  0.82362493,  0.87328337,  0.89459832,
                 1.11634592,  1.12325223,  1.12617243,  1.21597265,  1.22324724,
                 1.23938778,  1.28007182,  1.32340875,  1.3444802 ,  1.39667388,
                 1.41517789,  1.42733599,  1.46384562,  1.48753992,  1.49888065,
                 1.50855201,  1.51226656,  1.52159941,  1.54674525,  1.56452586,
                 1.56620677,  1.56800089,  1.56804683,  1.56325293,  1.55517349,
                 1.52862343,  1.52473884,  1.51651545,  1.51271117,  1.4759718 ,
                 1.44780469,  1.37909135,  1.37513737,  1.34926944,  1.34008171,
                 1.29082911,  1.27852725,  1.25925706,  1.21940475,  1.20510337,
                 1.19550691,  1.16776653,  0.89457483,  0.85682922,  0.77969179,
                 0.76803007,  0.72119979,  0.71162026,  0.70097437,  0.64666339,
                 0.64254993,  0.59595014,  0.59537079,  0.39284705,  0.30772041,
                 0.19944427,  0.11370244,  0.11239362,  0.09062844, -0.05118779,
                -0.25372646, -0.25753545, -0.28466944, -0.29167973, -0.32759939,
                -0.3603662 , -0.37474245, -0.38653161, -0.38945219, -0.39239248,
                -0.4257711 , -0.42898903, -0.43148438, -0.43186513, -0.40875729,
                -0.40199121, -0.39698397, -0.31201583, -0.25450812, -0.24495512,
                -0.13575062, -0.08753467, -0.07908995, -0.04889818,  0.00572697,
                 0.01553513,  0.07441225,  0.20701921,  0.24584939,  0.28064928,
                 0.2811854 ,  0.30891213,  0.34010958,  0.34945866,  0.47823857])

In [39]: poly9=PolynomialFeatures(degree=9)
         x2_poly9 = poly9.fit_transform(x2.reshape(-1,1))
         poly9.fit(x2.reshape(-1,1),y2)
         lin_reg.fit(x2_poly9,y2)

Out[39]: LinearRegression()

In [40]: plt.scatter(x2,y2,color='blue',facecolors = 'none',edgecolors = 'b', label = 'N=100')
         plt.plot(x2,y2, color= 'green', label = 'Y w.r.t. X')
         plt.plot(x2,lin_reg.predict(poly9.fit_transform(x2.reshape(-1,1))),color='red', label = 'M=
         9')
         plt.title('M=9')
         plt.legend()
         plt.grid(True)
         plt.show()
```



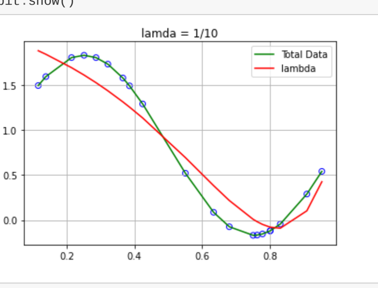
Now we will regularize using the sum of weights.

Draw chart for lamda is 1, 1/10, 1/100, 1/1000, 1/10000, 1/100000

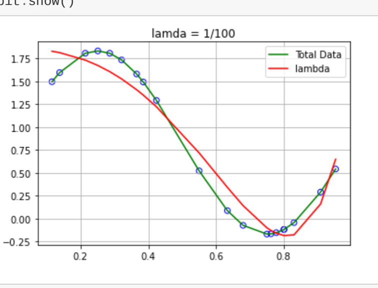
```
In [41]: poly_features = PolynomialFeatures(degree = 9)
         ridge = Ridge(alpha = 1)
         pip = Pipeline([('poly_features', poly_features),('ridge', ridge)])
         pip.fit(x.reshape(-1,1),y)
         a = pip.predict(x.reshape(-1,1))

         plt.scatter(x,y,color='blue',facecolors = 'none',edgecolors = 'b')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(x,a,color='red', label = 'lambda')
         plt.title('lamda = 1')
         plt.legend()
         plt.grid(True)
         plt.show()
```



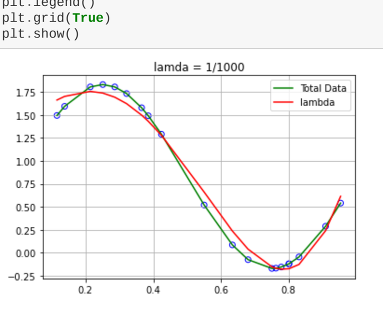

```
In [42]: poly_features = PolynomialFeatures(degree = 9)
         ridge = Ridge(alpha = 1/10)
         pip = Pipeline([('poly_features', poly_features),('ridge', ridge)])
         pip.fit(x.reshape(-1,1),y)
         a = pip.predict(x.reshape(-1,1))

         plt.scatter(x,y,color='blue',facecolors = 'none',edgecolors = 'b')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(x,a,color='red', label = 'lambda')
         plt.title('lamda = 1/10')
         plt.legend()
         plt.grid(True)
         plt.show()
```



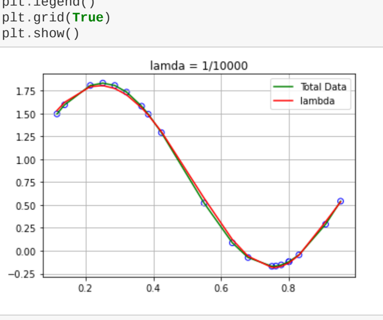

```
In [43]: poly_features = PolynomialFeatures(degree = 9)
         ridge = Ridge(alpha = 1/100)
         pip = Pipeline([('poly_features', poly_features),('ridge', ridge)])
         pip.fit(x.reshape(-1,1),y)
         a = pip.predict(x.reshape(-1,1))

         plt.scatter(x,y,color='blue',facecolors = 'none',edgecolors = 'b')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(x,a,color='red', label = 'lambda')
         plt.title('lamda = 1/100')
         plt.legend()
         plt.grid(True)
         plt.show()
```



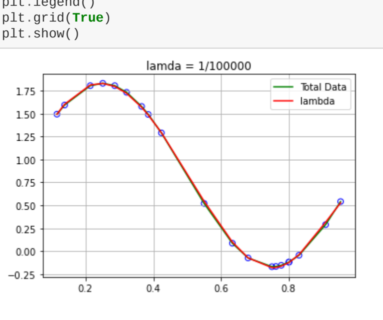

```
In [44]: poly_features = PolynomialFeatures(degree = 9)
         ridge = Ridge(alpha = 1/1000)
         pip = Pipeline([('poly_features', poly_features),('ridge', ridge)])
         pip.fit(x.reshape(-1,1),y)
         a = pip.predict(x.reshape(-1,1))

         plt.scatter(x,y,color='blue',facecolors = 'none',edgecolors = 'b')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(x,a,color='red', label = 'lambda')
         plt.title('lamda = 1/1000')
         plt.legend()
         plt.grid(True)
         plt.show()
```




```
In [45]: poly_features = PolynomialFeatures(degree = 9)
         ridge = Ridge(alpha = 1/10000)
         pip = Pipeline([('poly_features', poly_features),('ridge', ridge)])
         pip.fit(x.reshape(-1,1),y)
         a = pip.predict(x.reshape(-1,1))

         plt.scatter(x,y,color='blue',facecolors = 'none',edgecolors = 'b')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(x,a,color='red', label = 'lambda')
         plt.title('lamda = 1/10000')
         plt.legend()
         plt.grid(True)
         plt.show()
```

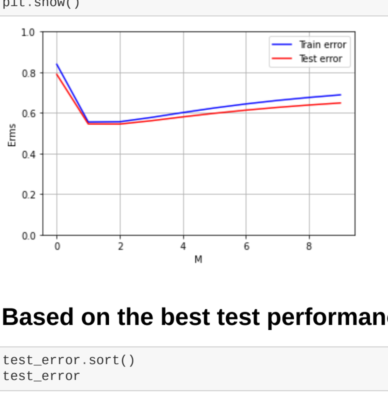



```
In [46]: poly_features = PolynomialFeatures(degree = 9)
         ridge = Ridge(alpha = 1/100000)
         pip = Pipeline([('poly_features', poly_features),('ridge', ridge)])
         pip.fit(x.reshape(-1,1),y)
         a = pip.predict(x.reshape(-1,1))

         plt.scatter(x,y,color='blue',facecolors = 'none',edgecolors = 'b')
         plt.plot(x,y, color= 'green', label = 'Total Data')
         plt.plot(x,a,color='red', label = 'lambda')
         plt.title('lamda = 1/100000')
         plt.legend()
         plt.grid(True)
         plt.show()
```



Now draw test and train error according to lamda

```
In [47]: lamda_train_error = np.empty(10)
         lamda_test_error = np.empty(10)

In [48]: for degree in range(10):
             est = make_pipeline(PolynomialFeatures(degree), Ridge(degree))
             est.fit(X_train_sorted.reshape(-1,1), Y_train_sorted)
             lamda_train_error[degree] = np.sqrt(mean_squared_error(Y_train_sorted, est.predict(X_tra
         in_sorted.reshape(-1,1))))
             lamda_test_error[degree] = np.sqrt(mean_squared_error(Y_test_sorted, est.predict(X_test_
         sorted.reshape(-1,1))))

In [49]: lamda_train_error

Out[49]: array([0.83916843, 0.55493622, 0.55626836, 0.57732065, 0.60148489,
                0.62404617, 0.6439136 , 0.66109274, 0.67589106, 0.6886647 ])

In [50]: lamda_test_error

Out[50]: array([0.78954853, 0.54530604, 0.54489509, 0.56117261, 0.58009811,
                0.59777665, 0.61334487, 0.62683649, 0.63851847, 0.64868345])

In [51]: plt.plot(np.arange(10), lamda_train_error, color='blue', label='Train error')
         plt.plot(np.arange(10), lamda_test_error, color='red', label='Test error')
         plt.ylim((0, 1))
         plt.ylabel('Erms')
         plt.xlabel('M')
         plt.legend()
         plt.grid(True)
         plt.show()
```



Based on the best test performance, what is your model?

```
In [52]: test_error.sort()
         test_error

Out[52]: array([0.34903408, 0.35017743, 0.36006323, 0.36104048, 0.36126409,
                0.41215655, 0.43283411, 0.46054198, 0.51347188, 0.83916843])

In [53]: lamda_test_error.sort()
         lamda_test_error

Out[53]: array([0.54489509, 0.54530604, 0.56117261, 0.58009811, 0.59777665,
                0.61334487, 0.62683649, 0.63851847, 0.64868345, 0.78954853])
```

Based on the above inferences Polynomial Regression of the degree 5 has the best performance. Hence is my model. But based on each iteration the model spits out a different best model.