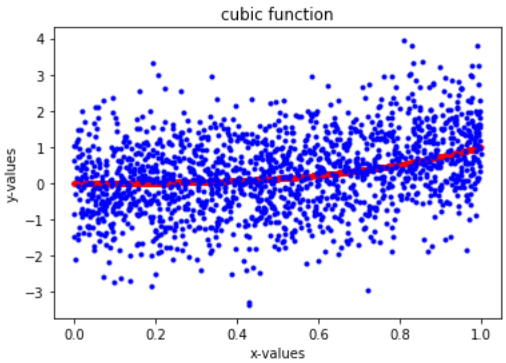
Lab Assignment 1

1.(a)

```
import numpy
numSamples = 1600
x= numpy.random.rand(numSamples, 1)
y = x**3
noise = numpy.random.randn(numSamples,1)
y_withNoise = y+noise

import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(x, y, "r.")
plt.plot(x, y_withNoise, "b.")
plt.xlabel('x-values')
plt.ylabel('y-values')
plt.title('cubic function')
plt.show()
```



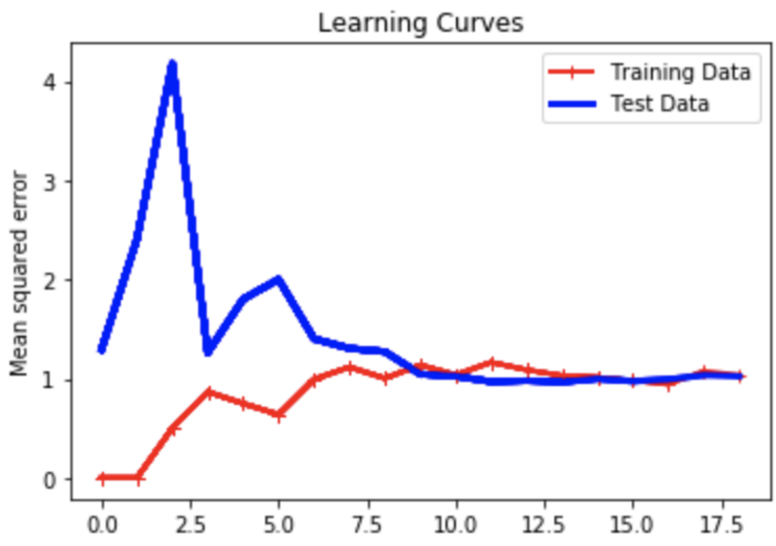
(b)

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y_withNoise, test_size = 0.2, random_state=42)
print("Number samples in training:", len(x_train))
print("Number samples in testing:", len(x_test))
```

```
In [3]: from sklearn.model_selection import train_test_split

        x_train, x_test, y_train, y_test = train_test_split(x, y_withNoise, test_size = 0.2, random_state=42)
        print("Number samples in training:", len(x_train))
        print("Number samples in testing:", len(x_test))

    Number samples in training: 1280
    Number samples in testing: 320
```

(c) **learning curves-linear regression models:**

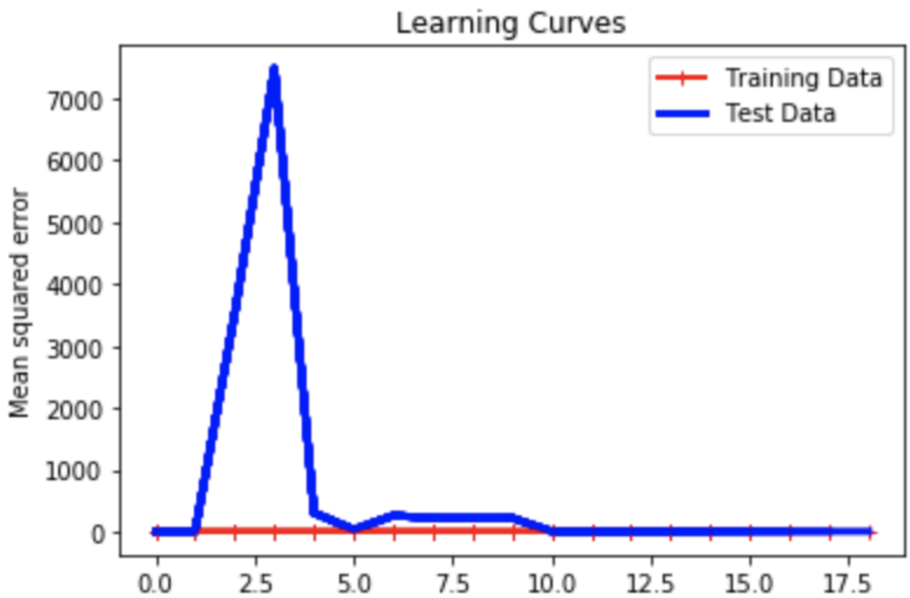
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
def plot_learning_curves(model, x, y):
    x_train, x_test, y_train, y_test = train_test_split(x, y_withNoise, test_size = 0.2,
  random_state=42)
    train_errors, test_errors= [], []
    for m in range(1, 20):
        model.fit(x_train[:m], y_train[:m])
        y_train_predict = model.predict(x_train[:m])
        y_test_predict = model.predict(x_test)
        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))
        test_errors.append(mean_squared_error(y_test_predict, y_test))
        plt.plot(train_errors, "r-+", linewidth =2, label = "Training Data")
        plt.plot(test_errors, "b-", linewidth = 3, label = "Test Data")
        plt.ylabel('Mean squared error')
        plt.title('Learning Curves')
        if m==1:
            plt.legend()

linear_reg_model = linear_model.LinearRegression()
plot_learning_curves(linear_reg_model, x, y_withNoise)
```



learning curves-polynomial regression models:

```
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree=5, include_bias = False)
x_poly = poly_features.fit_transform(x)
def plot_learning_curves(model, x, y):
    x_polyTrain, x_polyTest, y_polyTrain, y_polyTest = train_test_split(x_poly,
y_withNoise, test_size = 0.2, random_state = 42)
    train_errors, test_errors= [], []
    for m in range(1, 20):
        model.fit(x_polyTrain[:m], y_polyTrain[:m])
        y_polyTrain_predict = model.predict(x_polyTrain[:m])
        y_polyTest_predict = model.predict(x_polyTest)
        train_errors.append(mean_squared_error(y_polyTrain_predict,
y_polyTrain[:m]))
        test_errors.append(mean_squared_error(y_polyTest_predict,
y_polyTest))
        plt.plot(train_errors, "r-+", linewidth =2, label = "Training Data")
        plt.plot(test_errors, "b-", linewidth = 3, label = "Test Data")
        plt.ylabel('Mean squared error')
        plt.title('Learning Curves')
        if m==1:
            plt.legend()
```

```
poly_model = linear_model.LinearRegression()
plot_learning_curves(poly_model, x_poly, y_withNoise)
```



(d) In terms of these two models, linear regression model and polynomial regression model, I think polynomial regression model performs better. According to the learning curves, we can compare the mean squared error. The mean squared error of the linear regression model is almost one at the end. But the mean squared error of the polynomial regression model is almost zero. Thus, polynomial regression model performs better.

I think linear regression model is underfitting. The reason is that there exists mean squared error at the end. Furthermore, the performance of training data and test data are both poor.

When we increase the amount of training data on both models, we can find that the performance of models gradually becomes more stable and better.

2. (a)

```
from sklearn.datasets import load_diabetes
diabetes_data=load_diabetes()
diabetes_data.keys()
```

```
In [1]: from sklearn.datasets import load_diabetes

        diabetes_data=load_diabetes()
        diabetes_data.keys()

Out[1]: dict_keys(['data', 'target', 'DESCR', 'feature_names'])
```

(b)

from sklearn.model_selection import train_test_split

x_real = diabetes_data.data

y_real = diabetes_data.target

x_real_train, x_real_test, y_real_train, y_real_train = train_test_split(x_real, y_real, test_size = 0.2, random_state = 42)

print("Number samples in training:", len(x_real_train))

print("Number samples in testing:", len(x_real_test))

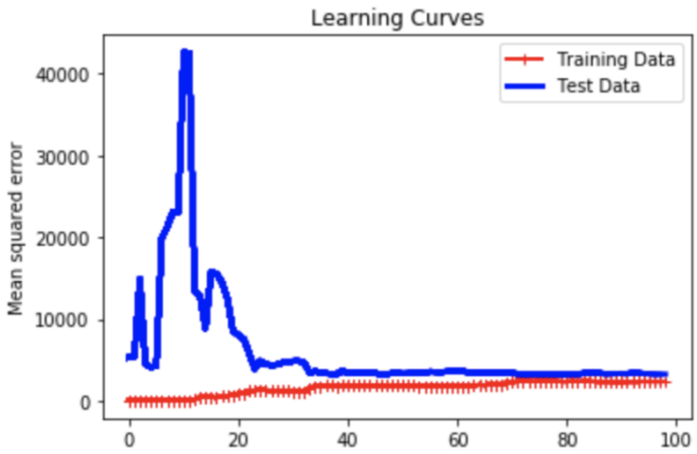
```
In [4]: from sklearn.model_selection import train_test_split

        x_real = diabetes_data.data
        y_real = diabetes_data.target
        x_real_train, x_real_test, y_real_train, y_real_train = train_test_split(x_real, y_real, test_size = 0.2, random_state = 42)
        print("Number samples in training:", len(x_real_train))
        print("Number samples in testing:", len(x_real_test))

        Number samples in training: 353
        Number samples in testing: 89
```

(c)

Linear regression model:

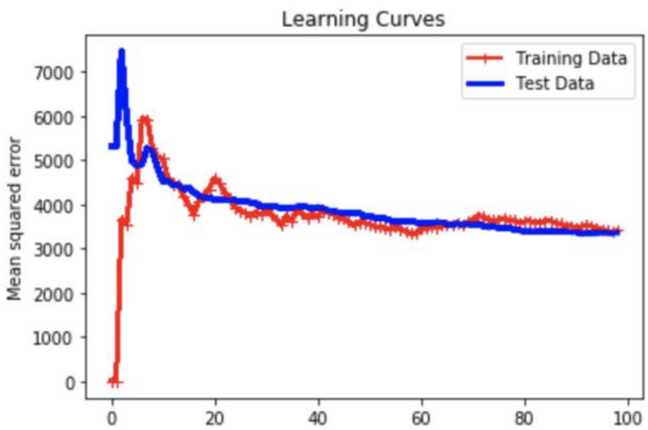
```
import matplotlib.pyplot as plt

from sklearn import linear_model

from sklearn.metrics import mean_squared_error


def plot_learning_curves(model, x, y):

    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state=42)

    train_errors, test_errors= [], []

    for m in range(1, 100):

        model.fit(x_train[:m], y_train[:m])

        y_train_predict = model.predict(x_train[:m])

        y_test_predict = model.predict(x_test)

        train_errors.append(mean_squared_error(y_train_predict, y_train[:m]))

        test_errors.append(mean_squared_error(y_test_predict, y_test))
```

```
        plt.plot(train_errors, "r-+", linewidth =2, label = "Training Data")
        plt.plot(test_errors, "b-", linewidth = 3, label = "Test Data")
        plt.ylabel('Mean squared error')
        plt.title('Learning Curves')
        if m ==1:
            plt.legend()

linear_reg_model=linear_model.LinearRegression()
plot_learning_curves(linear_reg_model, x_real, y_real)
```



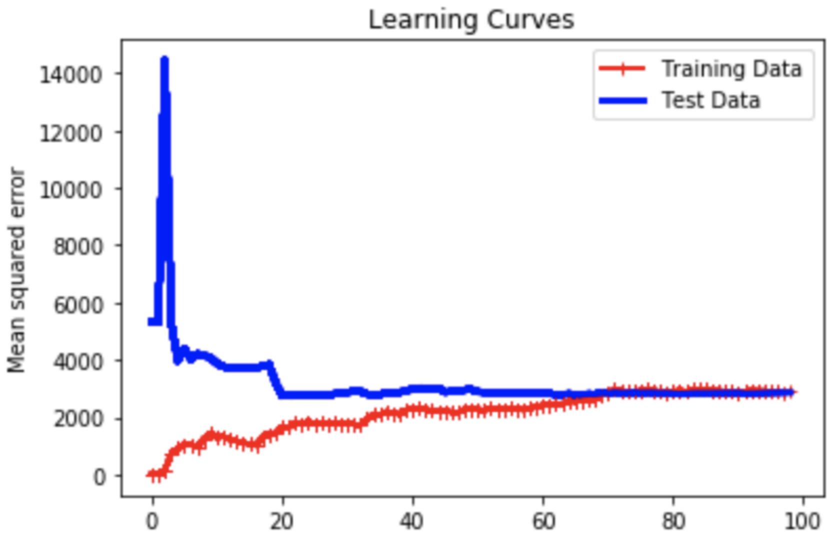
Ridge regression model:

```
ridge_model = linear_model.Ridge(alpha=0.5)
plot_learning_curves(ridge_model, x_real, y_real)
```



Lasso regression model:

```
lasso_model = linear_model.Lasso(alpha=0.5)
plot_learning_curves(lasso_model, x_real, y_real)
```



Polynomial regression model:

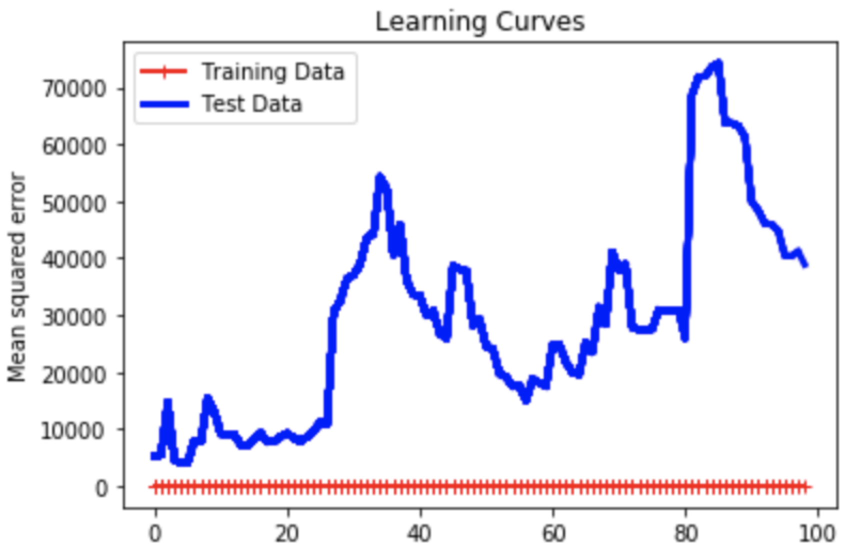
from sklearn.preprocessing import PolynomialFeatures

poly_features = PolynomialFeatures(degree=4, include_bias = False)

x_poly = poly_features.fit_transform(x_real)

poly_model = linear_model.LinearRegression()

plot_learning_curves(poly_model, x_poly, y_real)



(d)

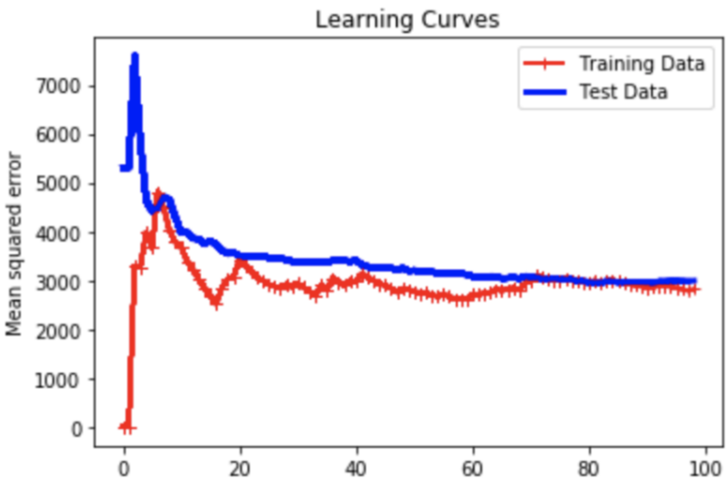
In terms of four regression models, I think the best model is the lasso regression model and the worst model is the polynomial regression model. According to the learning curves, we can find that the mean squared error is the lowest when the error becomes stable. But the polynomial model is overfitting. The mean squared error of training data is zero and the mean squared error of test data has changed constantly.

3. (a)

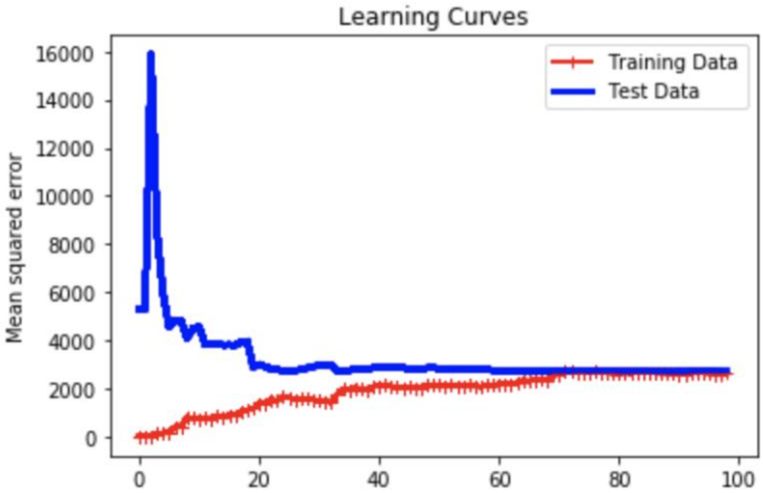
```
from sklearn.datasets import load_diabetes
diabetes_data=load_diabetes()
diabetes_data.keys()

from sklearn.model_selection import train_test_split
x_real = diabetes_data.data
y_real = diabetes_data.target
x_real_train, x_real_test, y_real_train, y_real_train = train_test_split(x_real, y_real, test_size = 0.2, random_state = 42)
```

(b)

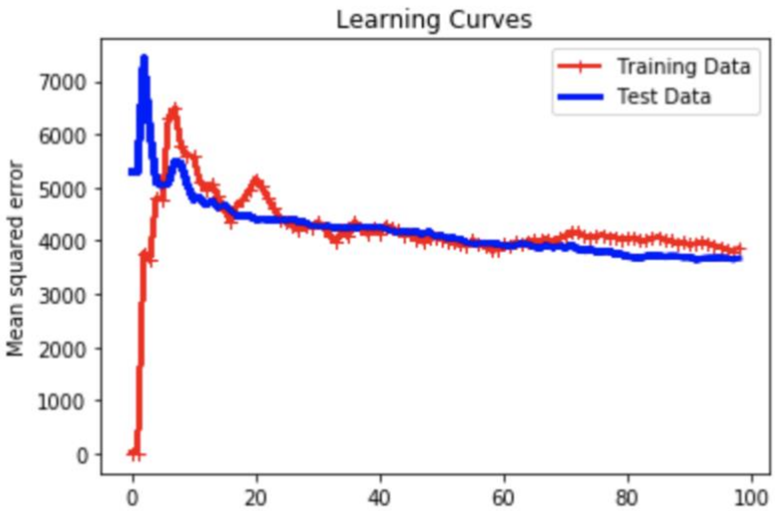
```
ridge_model = linear_model.Ridge(alpha=0.2)
plot_learning_curves(ridge_model, x_real, y_real)
```



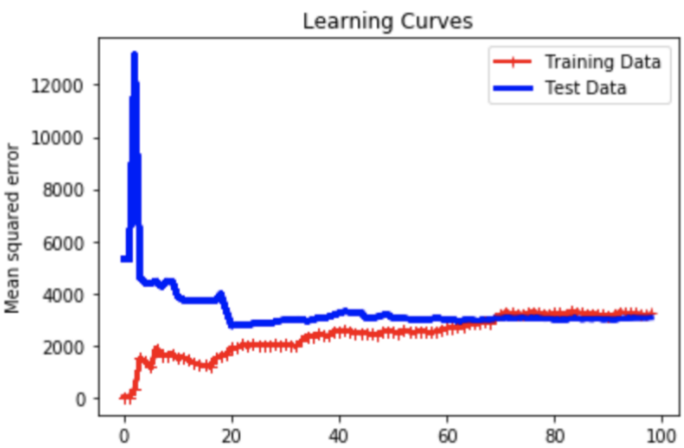

```
lasso_model = linear_model.Lasso(alpha=0.2)
plot_learning_curves(lasso_model, x_real, y_real)
```




```
ridge_model = linear_model.Ridge(alpha=0.8)
plot_learning_curves(ridge_model, x_real, y_real)
```




```
lasso_model = linear_model.Lasso(alpha=0.8)
plot_learning_curves(lasso_model, x_real, y_real)
```



(c) From the step 2, I plot the lasso and ridge regression model with regularization strength set to 0.5. At this step, I set regularization strength to 0.2 and 0.8 to see the impact of the regularization parameter. In terms of ridge regression model, when regularization parameter increases, the mean squared error also increases. About lasso regression model, the mean squared error does not have obvious impact.