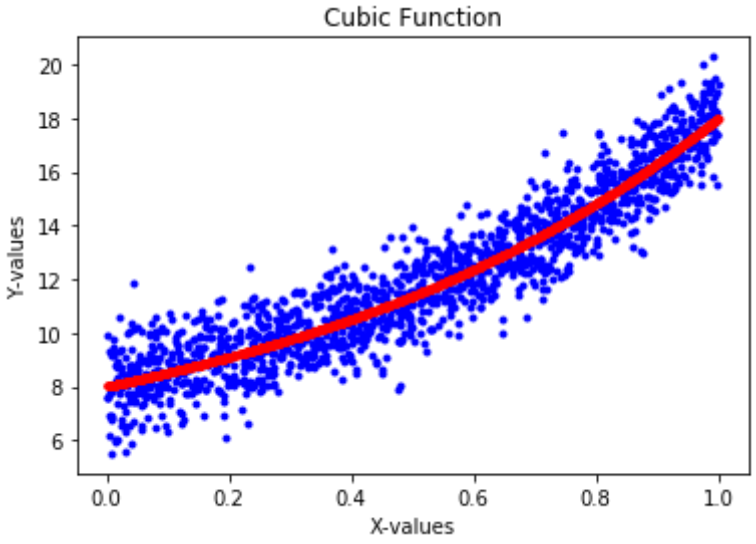
1. Synthetic Data: Comparative Analysis & Learning Curves

In [1]:
```
# Generate a dataset by sampling 1,600 values from a cubic function, wit
h noise added to the sampled values.
import numpy as np

numSamples = 1600
X = np.random.rand(numSamples, 1)
y = 3*(X**3) + 2*(X**2) + 5*X + 8
noise = np.random.randn(numSamples, 1)
y_withNoise = y + noise
```

In [2]:
```
# Visualize our dataset
import matplotlib.pyplot as plt

%matplotlib inline
plt.plot(X, y_withNoise, "b.")
plt.plot(X, y, "r.")
plt.xlabel("X-values")
plt.ylabel("Y-values")
plt.title("Cubic Function")
plt.show()
```



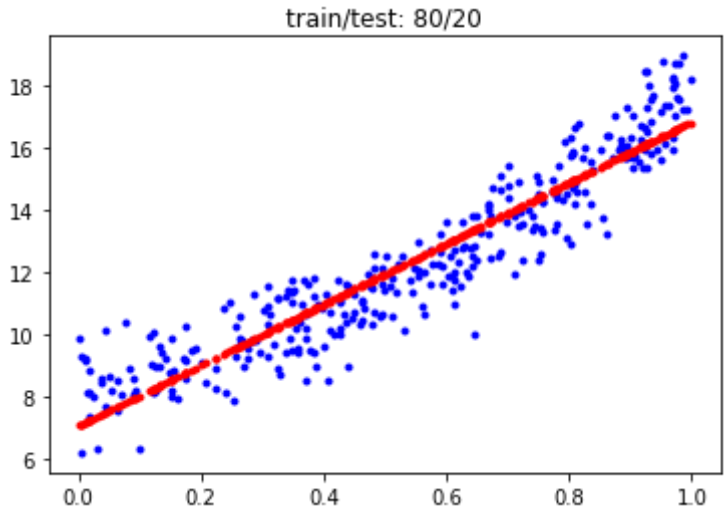
In [3]:
```
# Create a 80/20 train/test split of the dataset.
from sklearn.model_selection import train_test_split

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(y_train))
print("Number samples in testing:", len(y_test))
```

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

In [4]:
```
# Train linear regression model
from sklearn import linear_model

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Slope/Coefficient:", lr_model.coef_)
print("Intercept:", lr_model.intercept_)
```

```
Slope/Coefficient: [[9.76119326]]
Intercept: [7.05478595]
```

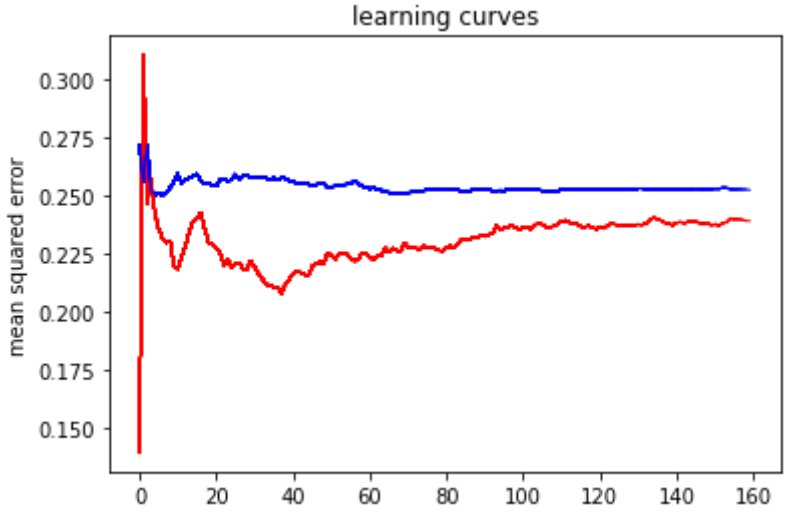
In [5]:
```
# Visualize predicted versus actual value for the test dataset
y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "b.")
plt.plot(X_test, y_predicted, "r.")
plt.title("train/test: 80/20")
plt.show()
```



In [6]:
```
# Evaluate model performance on the test dataset
from sklearn.metrics import mean_squared_error

mean_squared_error(y_test, y_predicted)
```

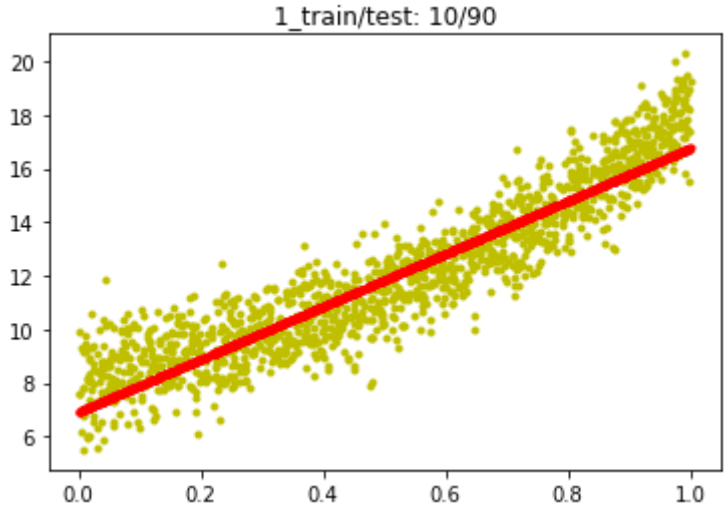
Out[6]:
```
1.08751082380476
```

In [8]:
```
# Plot learning curves
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(5, 1280, 8):
        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=1)
        plt.plot(test_errors, 'b-', linewidth=1)
        plt.ylabel("mean squared error")
        plt.title("learning curves")

linear_reg_model = linear_model.LinearRegression()
plot_learning_curves(linear_reg_model, X, y)
```



In [9]:
```
# Vary the amount of training data(10/90)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.9, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "* X + ", lr_mo
del.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("1_train/test: 10/90")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

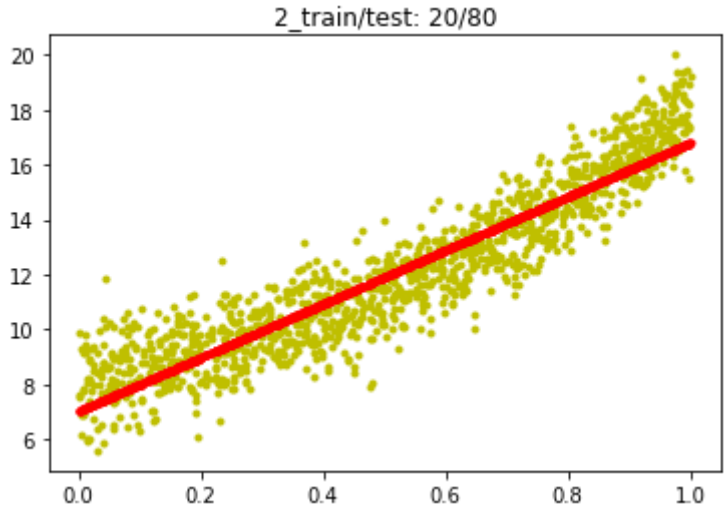
```
Number samples in training: 160
Number samples in testing: 1440
Linear regression equation: y = [[9.84118062]] * X +  [6.9261899]
```




```
Mean squared error: 1.2728288342163747
```

In [10]:
```
# Vary the amount of training data(20/80)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.8, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "* X + ", lr_mo
del.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("2_train/test: 20/80")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

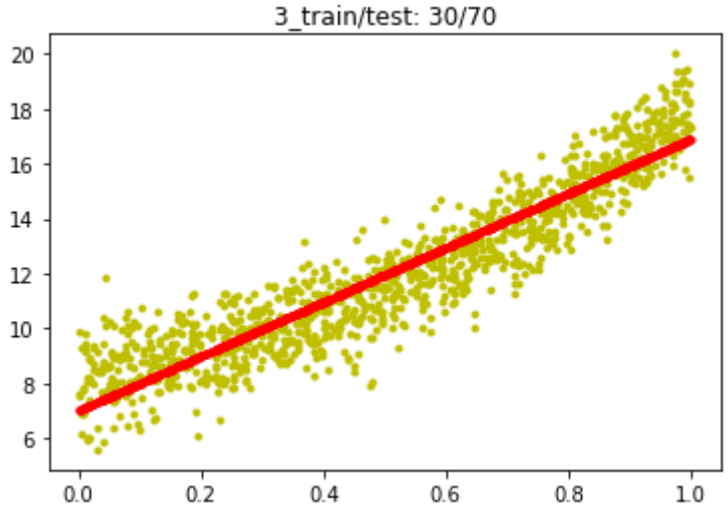
```
Number samples in training: 320
Number samples in testing: 1280
Linear regression equation: y = [[9.782097]] * X +  [7.02711738]
```




```
Mean squared error: 1.2528256896534786
```

In [11]:
```
# Vary the amount of training data(30/70)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.7, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "* X + ", lr_mo
del.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("3_train/test: 30/70")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

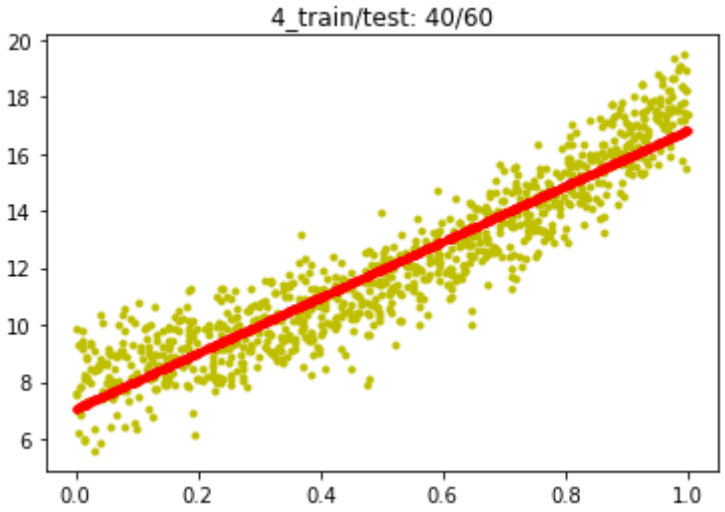
```
Number samples in training: 480
Number samples in testing: 1120
Linear regression equation: y = [[9.8951246]] * X +  [7.01611263]
```




```
Mean squared error: 1.2429124132800953
```

In [12]:
```
# Vary the amount of training data(40/60)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.6, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("4_train/test: 40/60")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

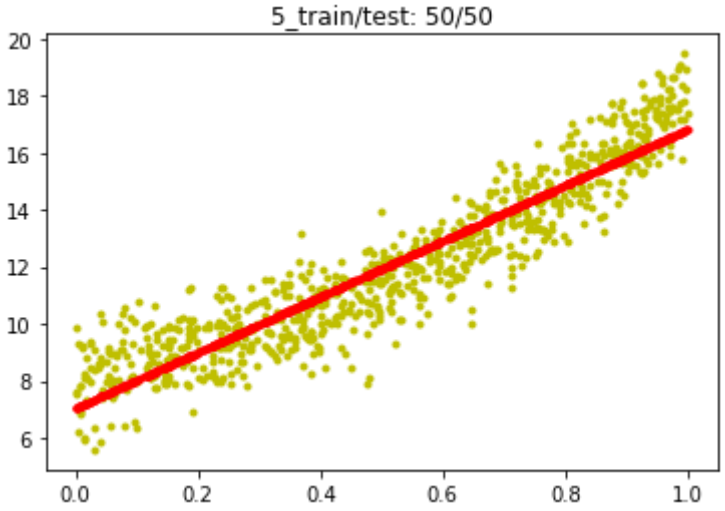
```
Number samples in training: 640
Number samples in testing: 960
Linear regression equation: y = [[9.7712408]] *X +  [7.06635813]
```




```
Mean squared error: 1.1721630515244679
```

In [13]:
```
# Vary the amount of training data(50/50)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.5, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("5_train/test: 50/50")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

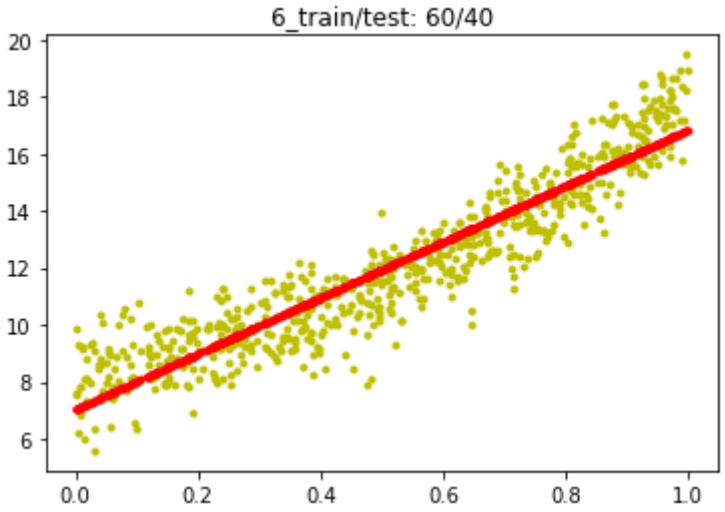
```
Number samples in training: 800
Number samples in testing: 800
Linear regression equation: y = [[9.77900875]] *X +  [7.04851176]
```




```
Mean squared error: 1.1975005010903021
```

In [14]:
```
# Vary the amount of training data(60/40)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.4, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("6_train/test: 60/40")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

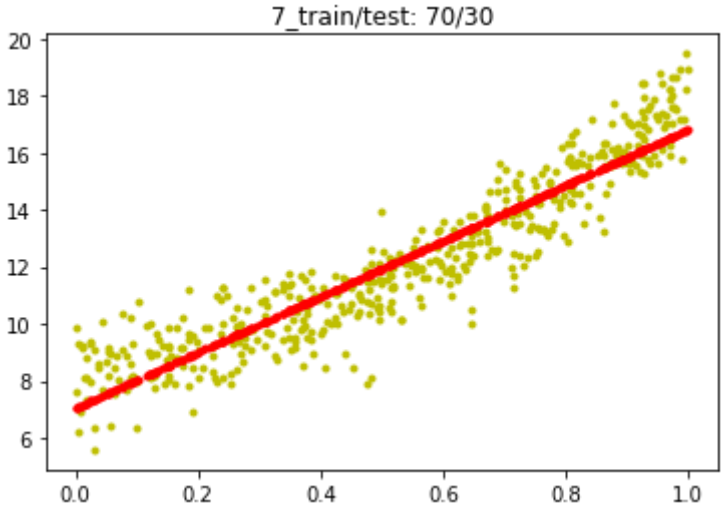
```
Number samples in training: 960
Number samples in testing: 640
Linear regression equation: y = [[9.80675329]] *X +  [7.03892537]
```




```
Mean squared error: 1.2104655518107608
```

In [15]:
```
# Vary the amount of training data(70/30)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.3, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("7_train/test: 70/30")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

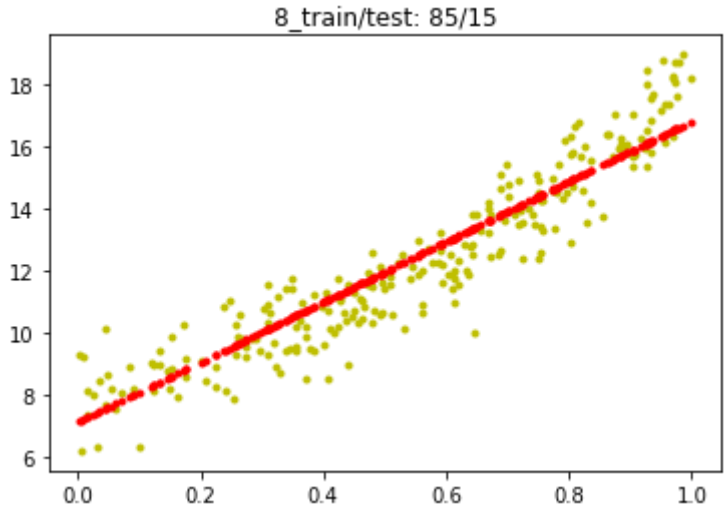
```
Number samples in training: 1120
Number samples in testing: 480
Linear regression equation: y = [[9.77080951]] *X +  [7.05596163]
```




```
Mean squared error: 1.2356618969348732
```

In [16]:
```
# Vary the amount of training data(85/15)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.15, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("8_train/test: 85/15")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

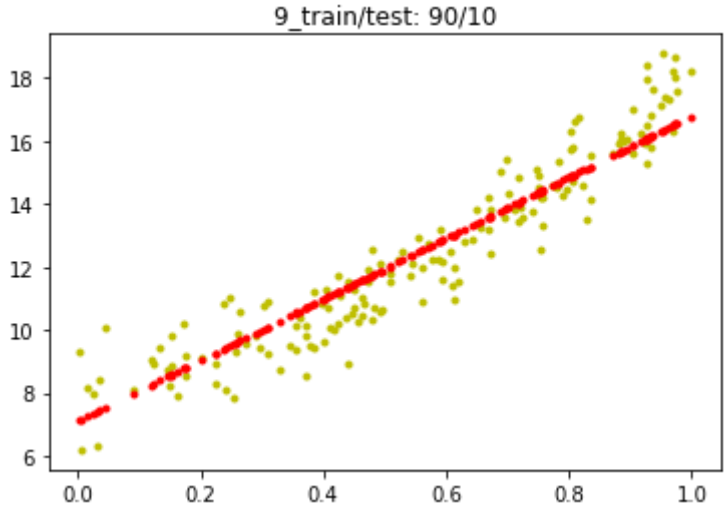
```
Number samples in training: 1360
Number samples in testing: 240
Linear regression equation: y = [[9.68964507]] *X +  [7.10538214]
```




```
Mean squared error: 1.0908458502767056
```

In [17]:
```
# Vary the amount of training data(90/10)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.10, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("9_train/test: 90/10")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

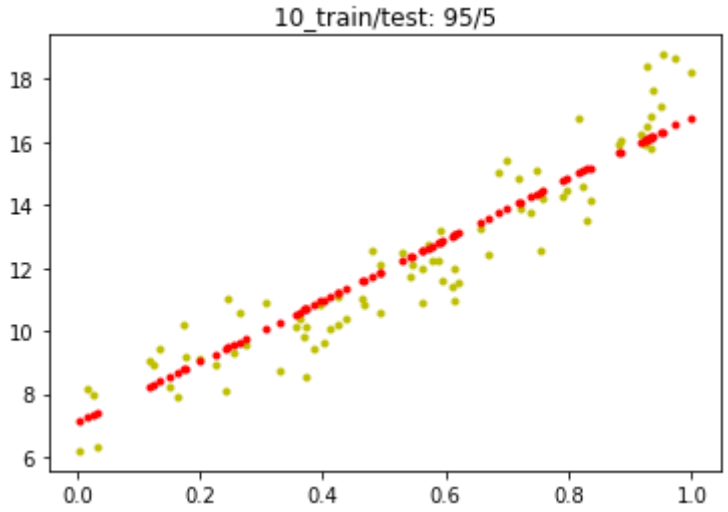
```
Number samples in training: 1440
Number samples in testing: 160
Linear regression equation: y = [[9.68094294]] *X +  [7.09966233]
```




```
Mean squared error: 0.9818794179606417
```

In [18]:
```
# Vary the amount of training data(95/5)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.05, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lr_model = linear_model.LinearRegression()
lr_model.fit(X_train, y_train)
print("Linear regression equation: y =", lr_model.coef_, "*X + ", lr_mod
el.intercept_)

y_predicted = lr_model.coef_*X_test + lr_model.intercept_

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("10_train/test: 95/5")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

```
Number samples in training: 1520
Number samples in testing: 80
Linear regression equation: y = [[9.70635949]] *X +  [7.08899212]
```




```
Mean squared error: 1.100979126247624
```

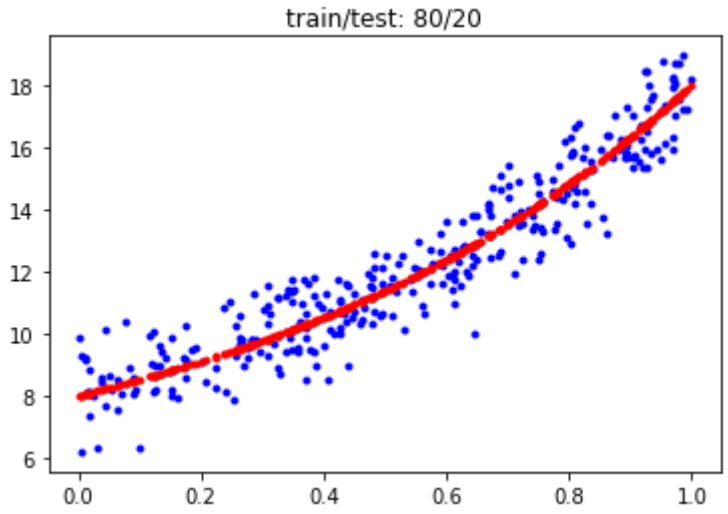
In [19]:
```
# Create a 80/20 train/test split of the dataset.
from sklearn.model_selection import train_test_split

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(y_train))
print("Number samples in testing:", len(y_test))
```

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

In [20]:
```
# Create Polynomial regression model
from sklearn.preprocessing import PolynomialFeatures

poly_reg = PolynomialFeatures(degree = 5, include_bias = False)
X_Poly = poly_reg.fit_transform(X)
print(X.shape)
print(X_Poly.shape)
```

```
(1600, 1)
(1600, 5)
```

In [21]:
```
lrmodel_2 = linear_model.LinearRegression()
lrmodel_2.fit(X_Poly, y)
y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

# Visualize
plt.plot(X_test, y_test, "b.")
plt.plot(X_test, y_predicted, "r.")
plt.title("train/test: 80/20")
plt.show()

# Evaluation
print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

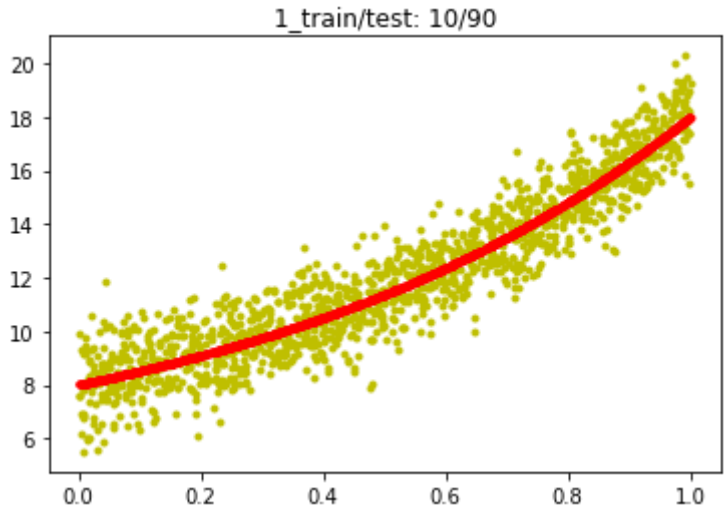



```
Mean squared error: 0.8170171127224108
```

In [35]:
```
# Plot learning curves
def plot_learning_curves(model, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, random_sta
te=42)
    train_errors, test_errors = [], []
    for m in range(5, 1280, 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=1)
        plt.plot(test_errors, 'b-', linewidth=1)
        plt.ylabel('Mean squared error')
        plt.title('Learning Curves')

plot_learning_curves(lrmodel_2, X_Poly, y)
```

In [36]:
```
# Vary the amount of training data(10/90)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.9, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5, include_bias = False)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("1_train/test: 10/90")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

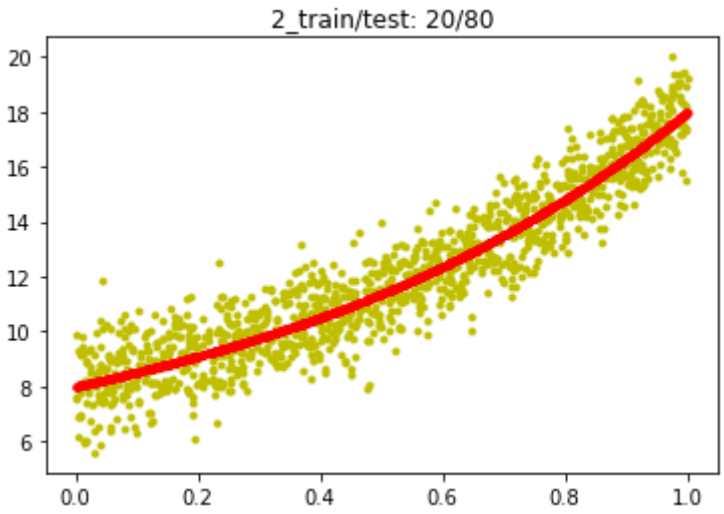
```
Number samples in training: 160
Number samples in testing: 1440
```




```
Mean squared error: 0.9865571244965742
```

In [37]:
```
# Vary the amount of training data(20/80)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.8, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5, include_bias = False)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("2_train/test: 20/80")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

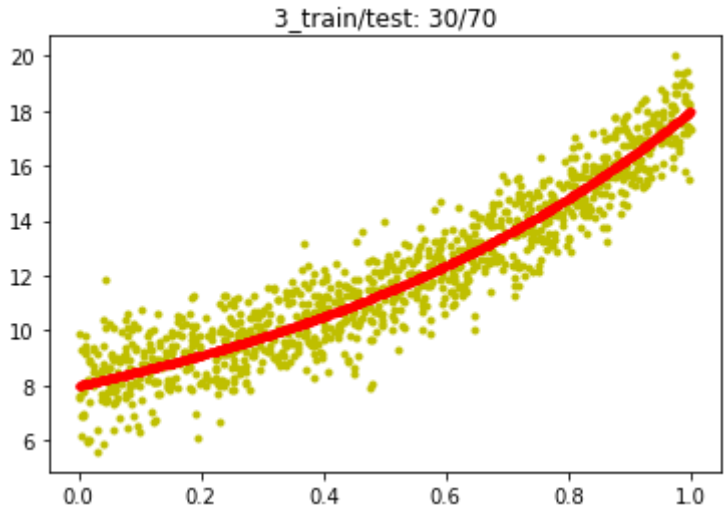
```
Number samples in training: 320
Number samples in testing: 1280
```




```
Mean squared error: 0.9717065996449639
```

In [38]:
```
# Vary the amount of training data(30/70)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.7, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5, include_bias = False)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("3_train/test: 30/70")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

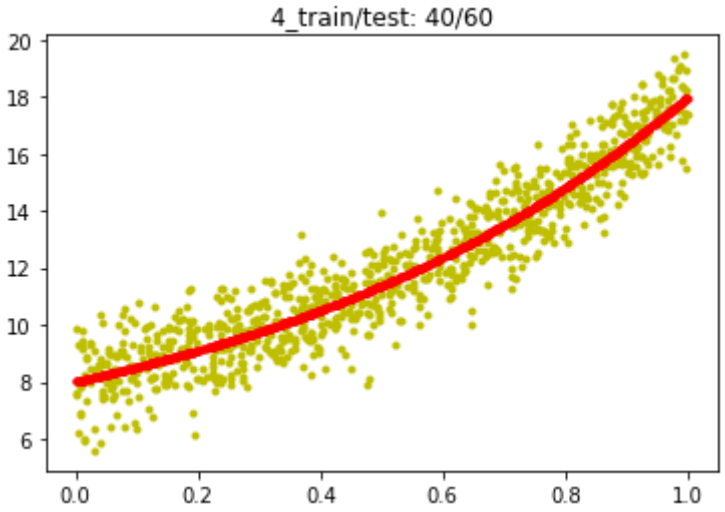
```
Number samples in training: 480
Number samples in testing: 1120
```




```
Mean squared error: 0.9554710434782366
```

In [39]:
```
# Vary the amount of training data(40/60)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.6, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("4_train/test: 40/60")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

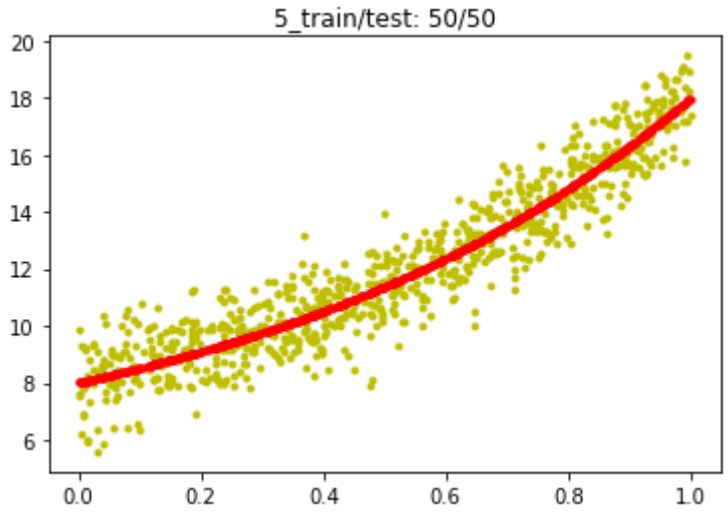
```
Number samples in training: 640
Number samples in testing: 960
```




```
Mean squared error: 0.9094392546352768
```

In [40]:
```
# Vary the amount of training data(50/50)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.5, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("5_train/test: 50/50")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

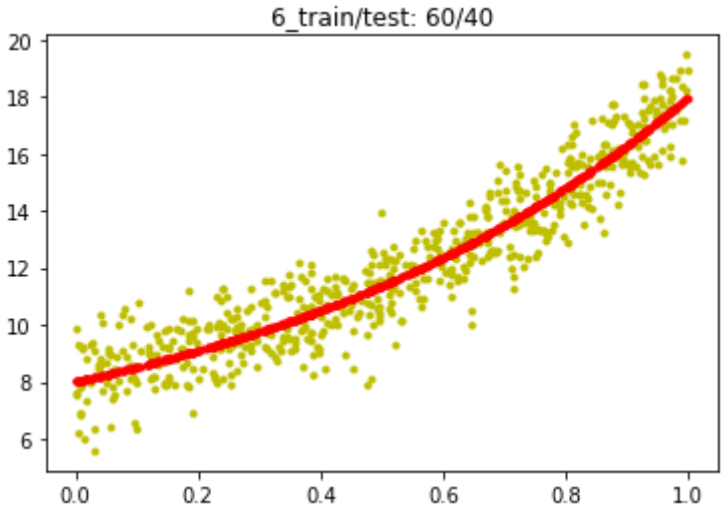
```
Number samples in training: 800
Number samples in testing: 800
```




```
Mean squared error: 0.9158323516243244
```

In [41]:
```
# Vary the amount of training data(60/40)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.4, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("6_train/test: 60/40")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

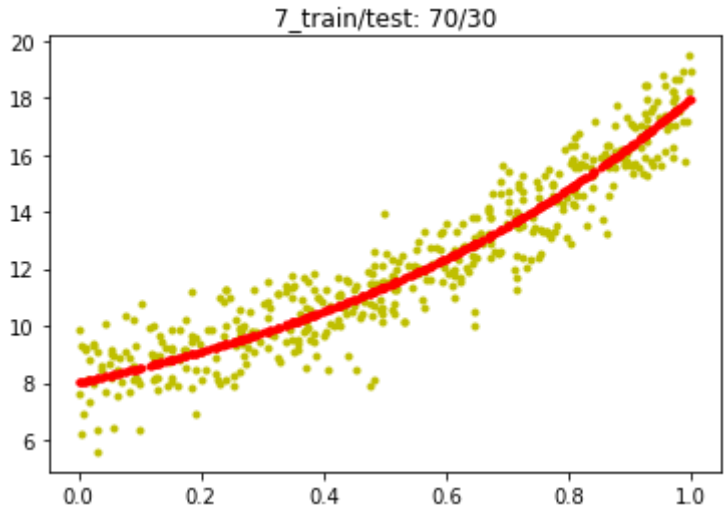
```
Number samples in training: 960
Number samples in testing: 640
```




```
Mean squared error: 0.9175990917206377
```

In [42]:
```
# Vary the amount of training data(70/30)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.3, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("7_train/test: 70/30")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

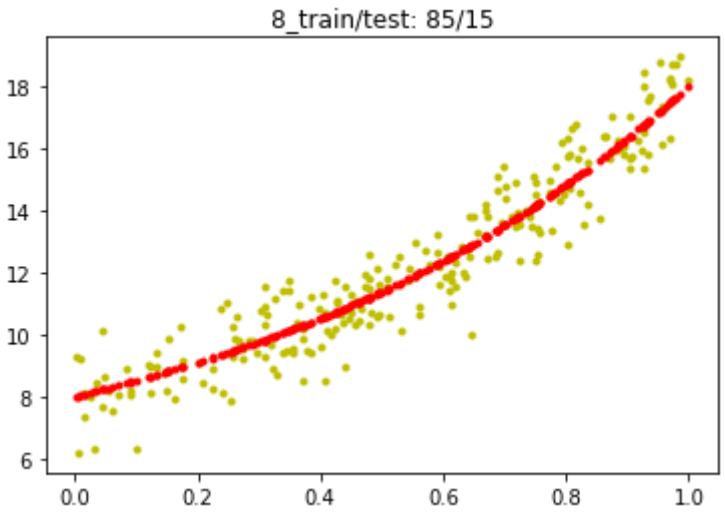
```
Number samples in training: 1120
Number samples in testing: 480
```




```
Mean squared error: 0.9298131389951872
```

In [43]:
```
# Vary the amount of training data(85/15)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.15, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("8_train/test: 85/15")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

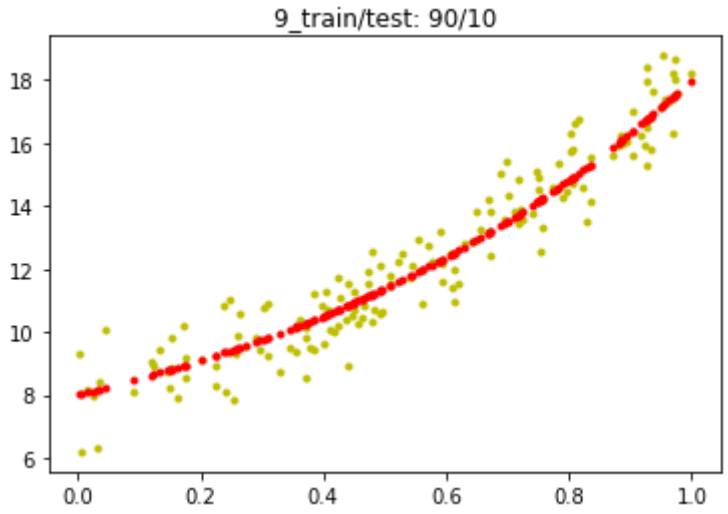
```
Number samples in training: 1360
Number samples in testing: 240
```




```
Mean squared error: 0.7880724018860163
```

In [44]:
```
# Vary the amount of training data(90/10)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.1, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("9_train/test: 90/10")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

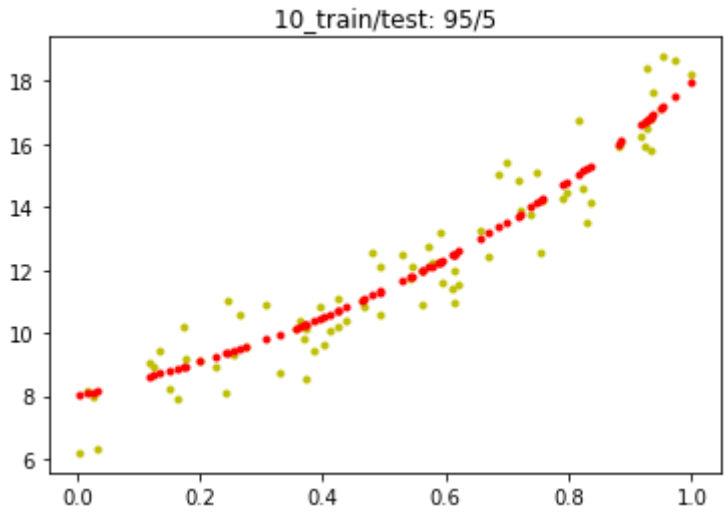
```
Number samples in training: 1440
Number samples in testing: 160
```




```
Mean squared error: 0.6984258188873694
```

In [45]:
```
# Vary the amount of training data(95/5)
X_train, X_test, y_train, y_test = train_test_split(X, y_withNoise, test
_size = 0.05, random_state = 42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))

lrmodel_2 = linear_model.LinearRegression()
poly_reg = PolynomialFeatures(degree = 5)
X_Poly = poly_reg.fit_transform(X)
lrmodel_2.fit(X_Poly, y)

y_predicted = lrmodel_2.predict(poly_reg.fit_transform(X_test))

plt.plot(X_test, y_test, "y.")
plt.plot(X_test, y_predicted, "r.")
plt.title("10_train/test: 95/5")
plt.show()

print("Mean squared error:", mean_squared_error(y_test, y_predicted))
```

```
Number samples in training: 1520
Number samples in testing: 80
```




```
Mean squared error: 0.8169345373700301
```

Write-up

The graph of linear regression model is a straight line. When split the the dataset into 80/20 train/test, the mean squared error is 1.239419177661406. When changing the weight of training data from 10/100 to 95/100, the mean squared errors are as follows:

1.2728288342163747,
1.2528256896534786,
1.2429124132800953,
1.1721630515244679,
1.1975005010903021,
1.2104655518107608,
1.2356618969348732,
1.0908458502767056,
0.9818794179606417,
1.100979126247624.

The graph of polynomial regression model is a curve. When split the the dataset into 80/20 train/test, the mean squared error is 1.0325832605937522. When changing the weight of training data from 10/100 to 95/100, the mean squared errors are as follows:

0.9865571244965742,
0.9717065996449639,
0.9554710434782366,
0.9094392546352768,
0.9158323516243244,
0.9175990917206377,
0.9298131389951872,
0.7880724018860163,
0.6984258188873694,
0.8169345373700301.

As we can see from the visualization, the mean squared error of polynomial regression model is smaller than the other one, and its graph is more similar to the original cubic function. That's how I reach the conclusion that polynomial regression model performs better than linear regression model.

I think the linear regression is underfitting and the polynomial regression fits well. Our data is generated from a cubic function, its graph is a curve not a straight line. In this case, the linear regression model is not that suitable.

When increasing the amount of training data, we can see the general trend of mean squared error decreases gradually on both models. It means we can improve the model's performance by increasing the amount of training data. However, comparing the mean squared error in two models, we can find that MSE is always lower in polynomial model. It means that although we can improve the performance by using more training data, the most fundamental thing is to choose the right model.

In [ ]:

2. Real Data: Comparative Analysis

In [1]:

```
# Load a real dataset
from sklearn.datasets import load_diabetes

diabetes_dataset = load_diabetes()
print(diabetes_dataset)
```

{'data': array([[0.03807591, 0.05068012, 0.06169621, ..., -0.0025922
6,
         0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
        -0.06832974, -0.09220405],
       [0.08529891, 0.05068012, 0.04445121, ..., -0.00259226,
         0.00286377, -0.02593034],
       ...,
       [0.04170844, 0.05068012, -0.01590626, ..., -0.01107952,
        -0.04687948, 0.01549073],
       [-0.04547248, -0.04464164, 0.03906215, ..., 0.02655962,
         0.04452837, -0.02593034],
       [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
        -0.00421986, 0.00306441]]), 'target': array([151., 75., 141.,
206., 135., 97., 138., 63., 110., 310., 101.,
        69., 179., 185., 118., 171., 166., 144., 97., 168., 68., 4
9.,
        68., 245., 184., 202., 137., 85., 131., 283., 129., 59., 34
1.,
        87., 65., 102., 265., 276., 252., 90., 100., 55., 61., 9
2.,
       259., 53., 190., 142., 75., 142., 155., 225., 59., 104., 18
2.,
       128., 52., 37., 170., 170., 61., 144., 52., 128., 71., 16
3.,
       150., 97., 160., 178., 48., 270., 202., 111., 85., 42., 17
0.,
       200., 252., 113., 143., 51., 52., 210., 65., 141., 55., 13
4.,
        42., 111., 98., 164., 48., 96., 90., 162., 150., 279., 9
2.,
        83., 128., 102., 302., 198., 95., 53., 134., 144., 232., 8
1.,
       104., 59., 246., 297., 258., 229., 275., 281., 179., 200., 20
0.,
       173., 180., 84., 121., 161., 99., 109., 115., 268., 274., 15
8.,
       107., 83., 103., 272., 85., 280., 336., 281., 118., 317., 23
5.,
        60., 174., 259., 178., 128., 96., 126., 288., 88., 292., 7
1.,
       197., 186., 25., 84., 96., 195., 53., 217., 172., 131., 21
4.,
        59., 70., 220., 268., 152., 47., 74., 295., 101., 151., 12
7.,
       237., 225., 81., 151., 107., 64., 138., 185., 265., 101., 13
7.,
       143., 141., 79., 292., 178., 91., 116., 86., 122., 72., 12
9.,
       142., 90., 158., 39., 196., 222., 277., 99., 196., 202., 15
5.,
        77., 191., 70., 73., 49., 65., 263., 248., 296., 214., 18
5.,
        78., 93., 252., 150., 77., 208., 77., 108., 160., 53., 22
0.,
       154., 259., 90., 246., 124., 67., 72., 257., 262., 275., 17
7.,

```
        71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  9
1.,
       150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 11
6.,
       145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  6
6.,
        94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 23
3.,
        60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  8
9.,
        31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 17
2.,
       114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 10
9.,
       191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 13
5.,
       244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 21
6.,
       263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  7
1.,
        77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 32
1.,
        58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 16
8.,
       140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  6
9.,
       219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 25
8.,
        43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  7
2.,
       140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  5
5.,
        84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 31
0.,
        94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 13
2.,
       220.,  57.]), 'DESCR': 'Diabetes dataset\n================\n\nNo
tes\n-----\n\nTen baseline variables, age, sex, body mass index, averag
e blood\npressure, and six blood serum measurements were obtained for e
ach of n =\n442 diabetes patients, as well as the response of interest,
a\nquantitative measure of disease progression one year after baselin
e.\n\nData Set Characteristics:\n\n  :Number of Instances: 442\n\n  :Nu
mber of Attributes: First 10 columns are numeric predictive values\n\n
:Target: Column 11 is a quantitative measure of disease progression one
year after baseline\n\n  :Attributes:\n    :Age:\n    :Sex:\n    :Body
mass index:\n    :Average blood pressure:\n    :S1:\n    :S2:\n    :S
3:\n    :S4:\n    :S5:\n    :S6:\n\nNote: Each of these 10 feature vari
ables have been mean centered and scaled by the standard deviation time
s `n_samples` (i.e. the sum of squares of each column totals 1).\n\nSou
rce URL:\nhttp://www4.stat.ncsu.edu/~boos/var.select/diabetes.html\n\nF
or more information see:\nBradley Efron, Trevor Hastie, Iain Johnstone
and Robert Tibshirani (2004) "Least Angle Regression," Annals of Statis
tics (with discussion), 407-499.\n(http://web.stanford.edu/~hastie/Pape
rs/LARS/LeastAngle_2002.pdf)\n', 'feature_names': ['age', 'sex', 'bmi',
'bp', 's1', 's2', 's3', 's4', 's5', 's6']}
```

In [2]:
```
diabetes_dataset.keys()
dir(diabetes_dataset)
print(diabetes_dataset.DESCR)
```

```
Diabetes dataset
================

Notes
-----

Ten baseline variables, age, sex, body mass index, average blood
pressure, and six blood serum measurements were obtained for each of n
=
442 diabetes patients, as well as the response of interest, a
quantitative measure of disease progression one year after baseline.

Data Set Characteristics:

  :Number of Instances: 442

  :Number of Attributes: First 10 columns are numeric predictive values

  :Target: Column 11 is a quantitative measure of disease progression o
ne year after baseline

  :Attributes:
    :Age:
    :Sex:
    :Body mass index:
    :Average blood pressure:
    :S1:
    :S2:
    :S3:
    :S4:
    :S5:
    :S6:

Note: Each of these 10 feature variables have been mean centered and sc
aled by the standard deviation times `n_samples` (i.e. the sum of squar
es of each column totals 1).

Source URL:
http://www4.stat.ncsu.edu/~boos/var.select/diabetes.html

For more information see:
Bradley Efron, Trevor Hastie, Iain Johnstone and Robert Tibshirani (200
4) "Least Angle Regression," Annals of Statistics (with discussion), 40
7-499.
(http://web.stanford.edu/~hastie/Papers/LARS/LeastAngle_2002.pdf)
```

In [3]:

```
# Create a 80/20 train/test split
from sklearn.model_selection import train_test_split

X = diabetes_dataset.data
y = diabetes_dataset.target
#print(X)
#print(y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
 random_state=42)
print("Number samples in training:", len(y_train))
print("Number samples in testing:", len(y_test))
```

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

In [4]:
```
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn import linear_model
def evaluation_error(model, X_train, y_train, X_test, y_test):
    model = model.fit(X_train, y_train)
    y_test_predict = model.predict(X_test)
    errors = mean_squared_error(y_test_predict, y_test)
    return errors

def train_error(model, X_train, y_train, X_test, y_test):
    model = model.fit(X_train, y_train)
    y_train_predict = model.predict(X_train)
    errors = mean_squared_error(y_train_predict, y_train)
    return errors


lr_model = linear_model.LinearRegression()
lr_error = train_error(lr_model, X_train, y_train, X_test, y_test)

ridge_model = linear_model.Ridge(alpha=0.5)
ridge_error = train_error(ridge_model, X_train, y_train, X_test, y_test)

lasso_model = linear_model.Lasso(alpha=0.5)
lasso_error = train_error(lasso_model, X_train, y_train, X_test, y_test)

poly_features = PolynomialFeatures(degree=4, include_bias = False)
X_train_poly = poly_features.fit_transform(X_train)
X_test_poly = poly_features.fit_transform(X_test)


poly_error = evaluation_error(lr_model, X_train_poly, y_train, X_test_po
ly, y_test)
poly_error_train = train_error(lr_model, X_train_poly, y_train, X_test_p
oly, y_test)

print("Linear Mean squared error", lr_error)
print("Ridge Mean squared error", ridge_error)
print("Lasso Mean squared error", lasso_error)
print("Polynomial Mean squared error", poly_error)
print("Polynomial train error", poly_error_train)
```

```
Linear Mean squared error 2868.546584216565
Ridge Mean squared error 3115.584377168976
Lasso Mean squared error 3271.22985697533
Polynomial Mean squared error 146911.30546507554
Polynomial train error 1.6221252788656565e-22
```

Write-up

Linear Mean squared error 2900.1732878832318
Ridge Mean squared error 2917.174211082088
Lasso Mean squared error 2945.147845137908
Polynomial Mean squared error 146911.30546507554

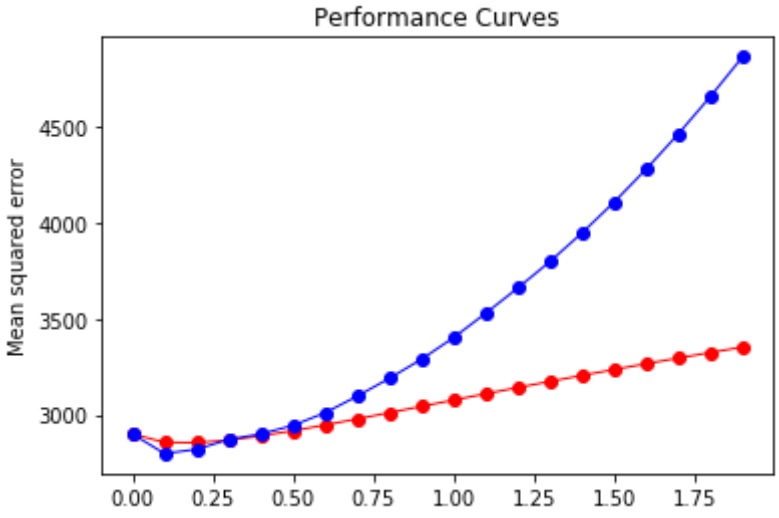
Among these four models, linear regression model performs best and polynomial model performs worst. Comparing the train MSE and evaluation MSE of polynomial model, we can find that the train MSE is small while the evaluation MSE is extremely big. The reason why polynomial model performs worst is that it's overfitting.

Linear regression performs better than ridge and lasso models here. According to graphs below, we can see that 0.5 is not the best regularization strength. When I try to use 0.1 as regularization strength, ridge and lasso models performs better than linear regression model.

3. Real Data: Tuning Hyperparameters for Regularized Models

In [5]:
```
import matplotlib.pyplot as plt

%matplotlib inline
alpha = 0

ridge_error, lasso_error, alpha_list = [], [], []
for m in range(20):

    alpha_list.append(alpha)
    ridge_model = linear_model.Ridge(alpha=alpha)
    lasso_model = linear_model.Lasso(alpha=alpha)
    ridge_error.append(evaluation_error(ridge_model, X_train, y_train, X
_test, y_test))
    lasso_error.append(evaluation_error(lasso_model, X_train, y_train, X
_test, y_test))
    alpha += 0.1

plt.plot(alpha_list, ridge_error, 'o-', color="r", linewidth=1, label="r
idge")
plt.plot(alpha_list, lasso_error, 'o-', color="b", linewidth=1, label="l
asso")
plt.ylabel('Mean squared error')
plt.title('Performance Curves')
plt.show()
```

```
/Users/ting/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.p
y:5: UserWarning: With alpha=0, this algorithm does not converge well.
You are advised to use the LinearRegression estimator
  """
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/
coordinate_descent.py:477: UserWarning: Coordinate descent with no regu
larization may lead to unexpected results and is discouraged.
  positive)
/Users/ting/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/
coordinate_descent.py:491: ConvergenceWarning: Objective did not conver
ge. You might want to increase the number of iterations. Fitting data w
ith very small alpha may cause precision problems.
  ConvergenceWarning)
```



Write-up

When varying the regularization strength, we can see that the mean squared error changes accordingly. It means that we can improve the performance by changing the reguralization parameter.

Ridge model performs better than lasso model. The red curve represents ridge model and the blue curve represents lasso model. From above graph, we can read that ridge MSE is smaller than lasso MSE in most cases.

In [ ]: