Encoding (Label encoding and One-hot encoding)

These are simple exercises useful for notebook 2

In [35]:

```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
```

Open 'grades.csv' and use appropriate encoding technique to convert to numerical data

In [2]:

```
df = pd.read_csv("grades.csv")
```

In [3]:

```
# write a function to do Label encoding
def label_encoding(column):
    encoding = {"S":10, "A":9, "B":8, "C":7, "D":6, "E":4, "F":0}
    Labeled_data = column.replace(encoding)
    return Labeled_data


df["Grades"] = label_encoding(df["Grades"])
df
```

Out[3]:

	Name	Grades
0	Nilesh	9
1	Jhon	10
2	James	7
3	Ravi	0
4	Sita	10
5	Raju	4
6	Praful	0
7	Ganga	8
8	Ram	9
9	Rahim	10
10	Julia	8

Open 'colors.csv' and use appropriate encoding technique to convert to numerical data

In [77]:

```
df = pd.read_csv("color.csv")
df
```

Out[77]:

	Objects	Color
0	Leaves	Green
1	Blood	Red
2	Sea	Blue
3	Rose	Red
4	Sun flower	Yellow
5	Apple	Red
6	Mango	Yellow
7	Banana	Yellow

Hint : you can achieve using pandas (no need for sklearn) check the documentation

In [78]:

```
# Answer
One_Hot_Encoded_Data = pd.get_dummies(df['Color'], prefix='Color')
df = df.drop(columns=['Color'])
df = pd.concat([df,One_Hot_Encoded_Data],axis=1)
df
```

Out[78]:

	Objects	Color_Blue	Color_Green	Color_Red	Color_Yellow
0	Leaves	0	1	0	0
1	Blood	0	0	1	0
2	Sea	1	0	0	0
3	Rose	0	0	1	0
4	Sun flower	0	0	0	1
5	Apple	0	0	1	0
6	Mango	0	0	0	1
7	Banana	0	0	0	1

Answer the following for both the data

- What are the categories in the data?
- what is the relationship between different categories in the grades column?
- Which encoding to be used Label encoding or one-hot encoding?

Note: If there is need of any assumption on grades assume grades follow pattern as it is in IIT M

What are the categories in the data?

For the grades data: There are seven different categories for the grades feature: S, A, B, C, D, E, F.

For the color data: There are four different colors: Blue, Green, Red and Yellow.

What is the relationship between different categories in the grades column?

Grades column has ordinal data where grades have an order between them. We used institute's grading system to assign levels to each grade:
{"S":10, "A":9, "B":8, "C":7, "D":6, "E":4, "F":0}

Which encoding to be used Label encoding or one-hot encoding?

For the grades data: Label encoding

For the color data: One-hot encoding

Regularized Least Squares

Ridge Regression

$$\hat{\theta} = argmin_{\theta}((y - X\theta)^{\top}(y - X\theta) + \lambda\theta^{\top}\theta)$$

Complete the following functions for ridge rigression using gradient descent (without sklearn)

- You can reuse the functions you have written in last week
- Also you need to change gradient and cost accordingly

Hint: If you use numpy for gradient calculation for theta, you may have to flatten the numpy array

In [6]:

```
def computeCost(X_appended, y, theta,lamda):
    """
    Compute cost for linear regression. Computes the cost of using theta as the
    parameter for linear regression to fit the data points in X and y.

    Parameters
    ----------
    X : with n features

    y : with (len(X),1)
    theta : array_like -> The parameters for the regression function. This is a vector
 of
        shape (n+1, 1).

    Returns
    -------
    J : float -> The value of the regression cost function.

    """
    # You need to return the following variables correctly
    m=X_appended.shape[0]
    J = np.dot((y-np.dot(X_appended,theta)).T, (y-np.dot(X_appended,theta))) + lamda*np
.dot(theta.T,theta)

    return J
```

In [7]:

```
def gradientDescent(X_appended, y, theta, alpha, lamda=10,max_iters= 1000):
    """
    Performs gradient descent to learn `theta`. Updates theta by taking `num_iters`
    gradient steps with learning rate `alpha`.

    Parameters
    ----------
    X : with n degree polynomial features

    y : array with shape of (len(X), 1)

    theta : array with shape of (n+1,1)

    alpha : float value, called as "learning rate"

    max_iters: maximum no of iteration

    Returns
    -------
    theta : array wih shape of (n,1),The learned linear regression parameters

    J_history : A python list for the values of the cost function after every iteratio
n. This is to check for convergence


    Hint:
    ------------
    1. Peform a single gradient step on the parameter vector theta.
    2. Loop over the number of iterations to update step by step.
    """
    # Initialize some useful values
    J_history = []
    J = computeCost(X_appended,y,theta,lamda)
    J_history.append(J)

    m = X_appended.shape[0]
    n = X_appended.shape[0] - 1
    count = 1
    while True:
        count+=1
        theta = (1-2*lamda*alpha)*theta - alpha*(1/m)*(np.dot(np.transpose(X_appended),
(np.dot(X_appended,theta) - y)))
        J = computeCost(X_appended,y,theta,lamda)
        J_history.append(J)
        cost = J

        if count>max_iters:
            break

    return theta, J_history
```

Load "dataWeek4Train.npz" for training the model and "dataWeek4Test.npz" for validating ridge rigression

Note: use np.load()

In [8]:

```
import numpy as np
from matplotlib import pyplot as plt
data = np.load("dataWeek4Train.npz")
data_test = np.load("dataWeek4Test.npz")
```

In [9]:

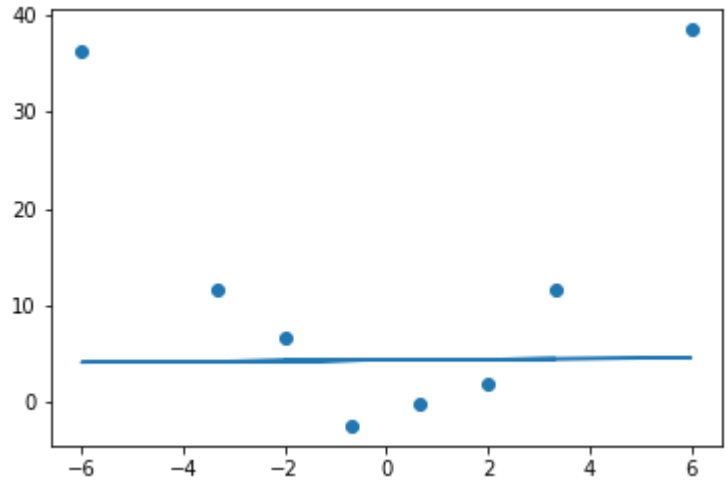
```
X_train = data['X_train']
y_train = data['y_train']


X_train_reshaped = np.reshape(X_train,(X_train.size,1))
y_train_reshaped = np.reshape(y_train,(y_train.size,1))
m=X_train_reshaped.shape[0]
X_train_appended = np.append(X_train_reshaped,np.ones((m,1)),axis=1)
```

In [10]:

```
theta = np.zeros((2,1))
theta, J_history = gradientDescent(X_train_appended, y_train_reshaped, theta, alpha=0.0
01, lamda=1,max_iters= 10000)
```

In [11]:

```
plt.scatter(X_train,y_train)
plt.plot(X_train_reshaped,np.dot(X_train_appended,theta))
plt.show();
```



Using functions written above try to fit the data with polynomial of degree 3 and

- plot the fit on train data, for λ = [0,100,10000]
- take learning rate (here it is alpha) 0.0001,
- max_iters as 1000 (you can change if you wish)
- Write comment how the fit changes as lambda and what is the reason?

In [12]:

```
# Write you code Here
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(3)
X_train_3 = poly.fit_transform(X_train_reshaped)
X_train_3
```

Out[12]:

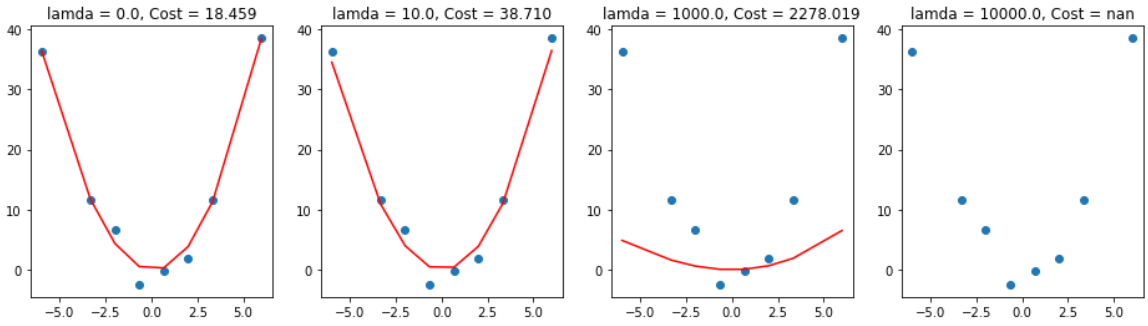
```
array([[   1.        ,    0.66666667,    0.44444444,    0.2962963 ],
       [   1.        ,   -6.        ,   36.        , -216.        ],
       [   1.        ,    3.33333333,   11.11111111,   37.03703704],
       [   1.        ,   -3.33333333,   11.11111111,  -37.03703704],
       [   1.        ,    6.        ,   36.        ,  216.        ],
       [   1.        ,   -0.66666667,    0.44444444,   -0.2962963 ],
       [   1.        ,   -2.        ,    4.        ,   -8.        ],
       [   1.        ,    2.        ,    4.        ,    8.        ]])
```

In [13]:

```
theta = np.zeros((4,1))
theta, J_history = gradientDescent(X_train_3, y_train_reshaped, theta, alpha=0.0001, la
mda=0,max_iters= 1000)
```

In [36]:

```
#Initializing multiple lamdas
lamdas = [0,10,1000,10000]


#Plotting degree 3 polynomial fit for different lamdas
fig,axes = plt.subplots(1,4,figsize=(16,4))
i=0

for lamda in lamdas:
    theta = np.zeros((4,1))
    theta, J_history = gradientDescent(X_train_3, y_train_reshaped, theta, alpha=0.0001
, lamda = lamda,max_iters= 2000)

    y_pred_3 = np.dot(X_train_3,theta)
    xs, ys = zip(*sorted(zip(X_train_reshaped, y_pred_3)))

    axes[i].scatter(X_train,y_train)
    axes[i].plot(xs,ys,'r')
    axes[i].set_title('lamda = %.1f, Cost = %.3f' %(lamda,computeCost(X_train_3,y_train
_reshaped,theta,lamda=lamda)))
    i+=1

plt.show();
```



Use same data "dataWeek4Train.npz" for training the model and "dataWeek4Test.npz" for validating the model for all three models4

In the following exercises you can use inbuilt regressionn functions from sklearn

The cost function explodes for very large lamda = 10000.

As lambda increases, the fitted model becomes less complex and has a reduced fit on the training data.

Linear Regression (Recap)

Write a generic function decribed in the following cell which takes in data, regression type - lasso,ridge and usual linear , degree of the polynomial and then the alpha values (which are regularization parameters - lambda λ , in sklearn they are named as alpha so its just a notaitonal difference).

Hint: Use PolynomialFeatures from sklearn and set bias to Flase

Use 9 degree polynomial features to fit the data using Linear Rigression. Comment on the fit of the model (Overfit, underfit, perfect fit) and also plot prediction on training data along with ground truth

In [32]:

```
''' Import neccessay packages '''
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.metrics import mean_squared_error


def regression(X,Y,reg_type ="ridge",degree = 9,alpha=0.1, max_iter=10000):
    '''
    Write a function which takes data X,Y
    Type of regression - usual linear , ridge , lasso
    Degree of polynomial features - here it is 10
    Regularization parameter alpha
    Return predicted output Y hat, Mean square error , regression object
    '''
    poly = PolynomialFeatures(degree, include_bias=False)
    x2 = poly.fit_transform(X)

    if reg_type == 'linear':
        Model = LinearRegression().fit(x2,Y)

    if reg_type == 'ridge':
        Model = Ridge(alpha=alpha, max_iter=max_iter)
        Model.fit(x2,Y)

    if reg_type == 'lasso':
        Model = Lasso(alpha=alpha, max_iter=max_iter)
        Model.fit(x2,Y)

    Y_hat = Model.predict(x2)
    mse = mean_squared_error(Y,Y_hat)

    reg = Model

    return Y_hat, mse , reg
```

Ridge Rigression and Lasso for polynomial fitting with degree 9

In previous you observed that how the model has fit the training data, now try the same thing using regularization both Ridge and Lasso. Use the generic function that you wrote in previous cell.

- Now use polynomial regression of order 9 along with Ridge regression and Lasso regression on "dataWeek4Train.npz" for $\lambda = [0, 1, 10^5, 10^{10}]$ and plot the true vs predicted values for each λ (Note here that in sklearn lambda is alpha)
- Plot the prediction on training data along with ground truth for various values of λ for both ridge and lasso. And also print the coefficients of model for each λ
- Also plot trainning error vs λ

Note : Here you can use ridge Rigression from sklearn and also set bias to False in polynomial features

- You can use library to generate polynomial features

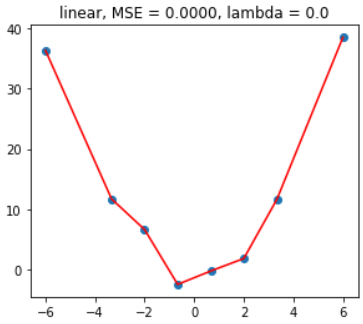
Lasso Regression

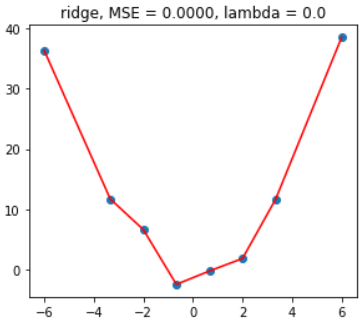
$$\hat{\theta} = argmin_\theta(\frac{1}{2}\sum_{i=1}^{N}(y_i - \theta_0 - \sum_{j=1}^{p}x_{ij}\theta_j)^2 + \lambda\sum_{j=1}^{p}|\theta_j|)$$

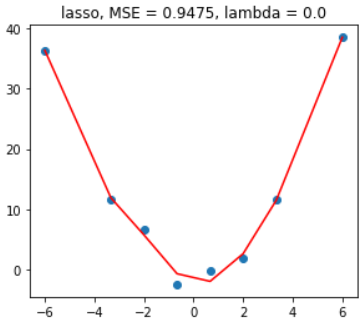
In [46]:

```
# Write your code here, plot the prediction on the train data, and print the coefficien
ts learned for ridge and lasso
# You can combine both Ridge and Lasso, But for Train error vs Lambda, plot them sepera
tely for lasso and ridge
# Plotting and calling the function for multiple lamdas and different types of regressi
on
lamdas = [0,1,10**5,10**10]

for lamda in lamdas:
    fig,axes = plt.subplots(1,3,figsize=(16,4))
    i=0
    reg_types = ['linear','ridge','lasso']

    for reg_type in reg_types:

        Y_hat, mse , reg = regression(X_train_reshaped,y_train_reshaped,reg_type =reg_t
ype,degree = 9,alpha=lamda)

        print('\nModel coefficients for reg_type = %s, lambda = %d:\n' %(reg_type,lamda
), reg.coef_)
        xs, ys = zip(*sorted(zip(X_train_reshaped, Y_hat)))

        axes[i].scatter(X_train,y_train)
        axes[i].plot(xs,ys,'r')
        axes[i].set_title('%s, MSE = %.4f, lambda = %.1f' %(reg_type,mse,lamda))
        i+=1

    plt.show();
```

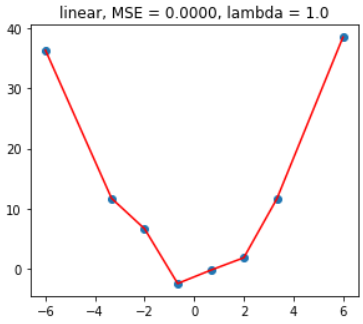
```
Model coefficients for reg_type = linear, lambda = 0:
 [[ 1.55313416e+00  1.60950532e-01  4.42676378e-01  4.86390666e-01
  -4.54833769e-01 -4.55957119e-02  4.68974250e-02  9.10972294e-04
  -9.62059801e-04]]

Model coefficients for reg_type = ridge, lambda = 0:
 [[ 1.55315827e+00  1.60951239e-01  4.42683559e-01  4.86392799e-01
  -4.54839905e-01 -4.55959576e-02  4.68980382e-02  9.10977519e-04
  -9.62072268e-04]]

Model coefficients for reg_type = lasso, lambda = 0:
 [-9.61891934e-01  1.80155618e+00  2.12126541e-02 -6.01342106e-02
   6.97543481e-03  7.54273760e-04 -1.24225762e-04  1.03651390e-05
  -1.70620916e-06]
```

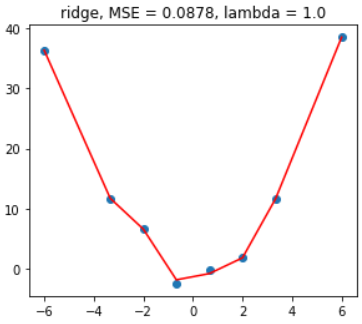


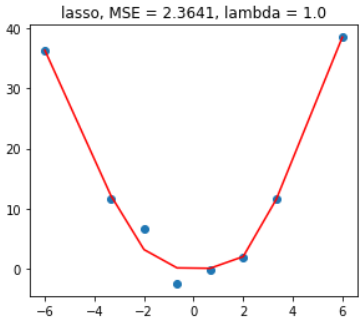




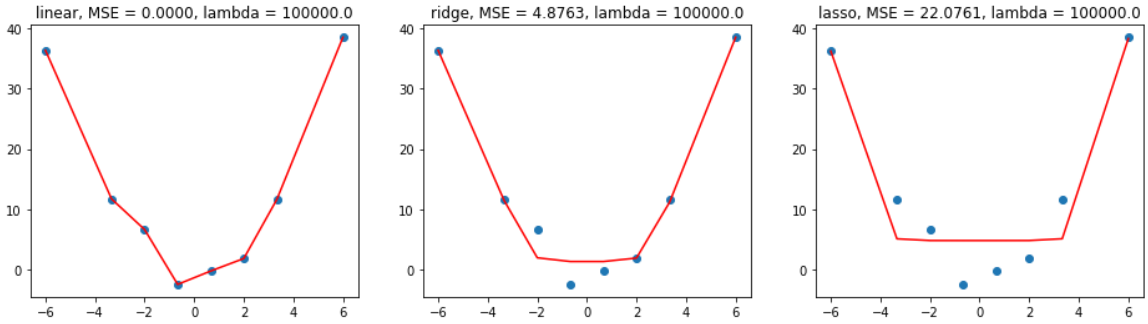

```
Model coefficients for reg_type = linear, lambda = 1:
 [[ 1.55313416e+00  1.60950532e-01  4.42676378e-01  4.86390666e-01
  -4.54833769e-01 -4.55957119e-02  4.68974250e-02  9.10972294e-04
  -9.62059801e-04]]

Model coefficients for reg_type = ridge, lambda = 1:
 [[ 7.44116435e-01  1.59227497e-01  1.92021948e-01  4.81193528e-01
  -2.78416132e-01 -4.49908395e-02  2.98148846e-02  8.98201036e-04
  -6.17815448e-04]]

Model coefficients for reg_type = lasso, lambda = 1:
 [-0.00000000e+00  3.61386328e-01 -1.16797299e-01  8.39353856e-02
   1.15032998e-02 -1.79891192e-03 -1.27622543e-04 -2.24084769e-07
  -2.72018086e-06]
```



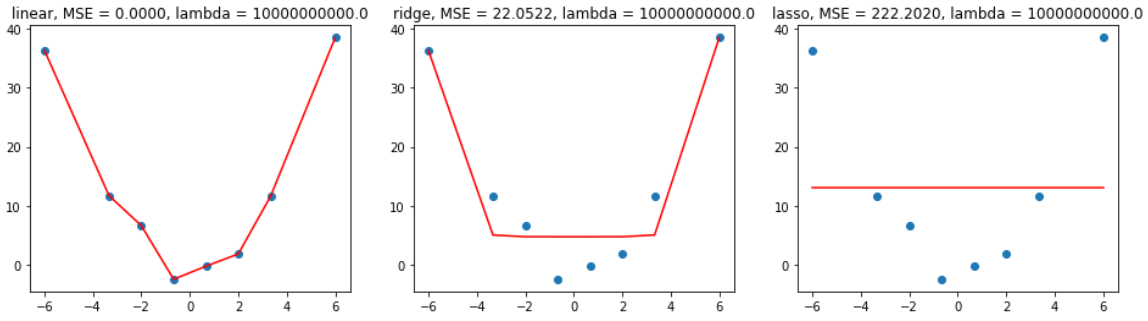




```
Model coefficients for reg_type = linear, lambda = 100000:
 [[ 1.55313416e+00  1.60950532e-01  4.42676378e-01  4.86390666e-01
  -4.54833769e-01 -4.55957119e-02  4.68974250e-02  9.10972294e-04
  -9.62059801e-04]]

Model coefficients for reg_type = ridge, lambda = 100000:
 [[-7.44481536e-05  2.43355499e-04 -3.10644737e-04  1.58106688e-03
  -8.54353734e-04  9.95073541e-03  6.17908868e-05 -2.56149945e-04
  -9.43202852e-07]]

Model coefficients for reg_type = lasso, lambda = 100000:
 [-0.00000000e+00  0.00000000e+00 -0.00000000e+00  0.00000000e+00
   0.00000000e+00  0.00000000e+00  0.00000000e+00  1.93067649e-05
   1.03278897e-07]
```

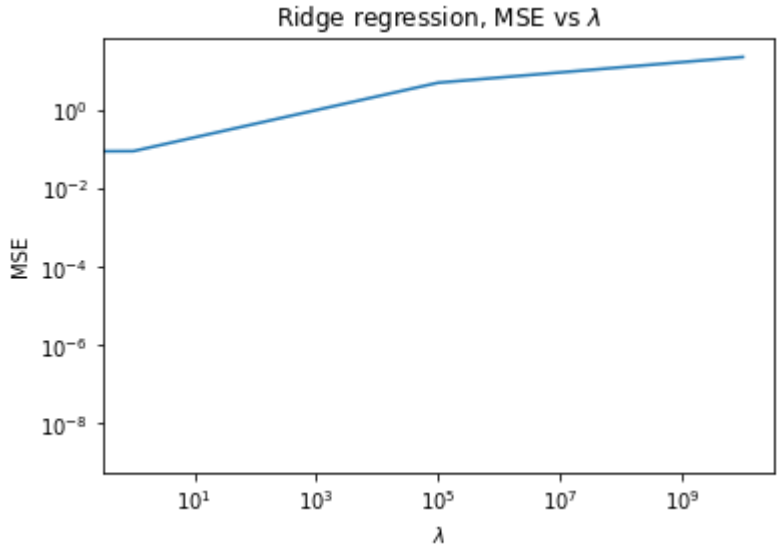


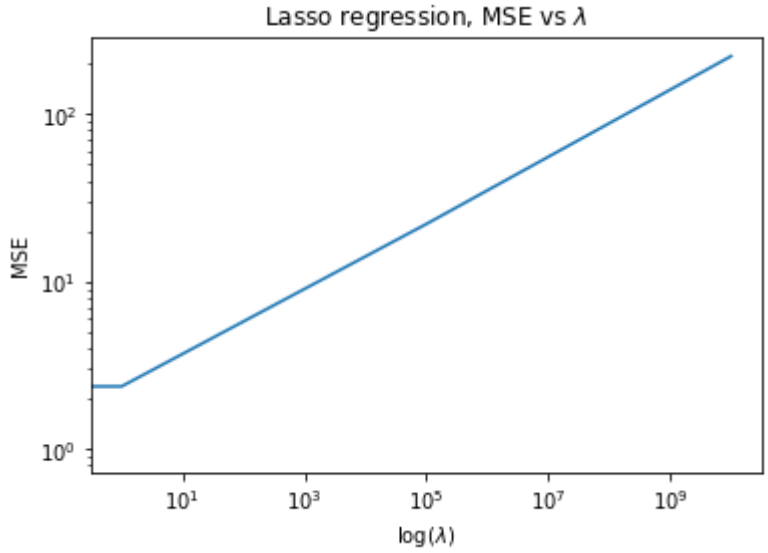

```
Model coefficients for reg_type = linear, lambda = 10000000000:
 [[ 1.55313416e+00  1.60950532e-01  4.42676378e-01  4.86390666e-01
  -4.54833769e-01 -4.55957119e-02  4.68974250e-02  9.10972294e-04
  -9.62059801e-04]]

Model coefficients for reg_type = ridge, lambda = 10000000000:
 [[-8.41944511e-10  1.37336483e-08 -4.10293246e-09  1.60946222e-07
  -1.85729179e-08  1.78835737e-06 -8.15144653e-08  1.94013952e-05
   1.09490809e-07]]

Model coefficients for reg_type = lasso, lambda = 10000000000:
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
```



In [66]:

```
#Training error vs lambda:
#for ridge:
mse_history=[]
for lamda in lamdas:
    Y_hat, mse , reg = regression(X_train_reshaped, y_train_reshaped, reg_type ='ridge'
, degree = 9, alpha=lamda)
    mse_history.append(mse)
plt.loglog(lamdas,mse_history)
plt.xlabel(r'$\lambda $')
plt.ylabel('MSE')
plt.title('Ridge regression, MSE vs ' + r'$\lambda $')
plt.show();

#for lasso:
mse_history=[]
for lamda in lamdas:
    Y_hat, mse , reg = regression(X_train_reshaped, y_train_reshaped, reg_type ='lasso'
, degree = 9, alpha=lamda)
    mse_history.append(mse)
plt.loglog(lamdas,mse_history)
plt.xlabel('log(' + r'$\lambda $'+')')
plt.ylabel('MSE')
plt.title('Lasso regression, MSE vs ' + r'$\lambda $')
plt.show();
```





Ridge Rigression and Lasso for polynomial fitting to predict on validation set

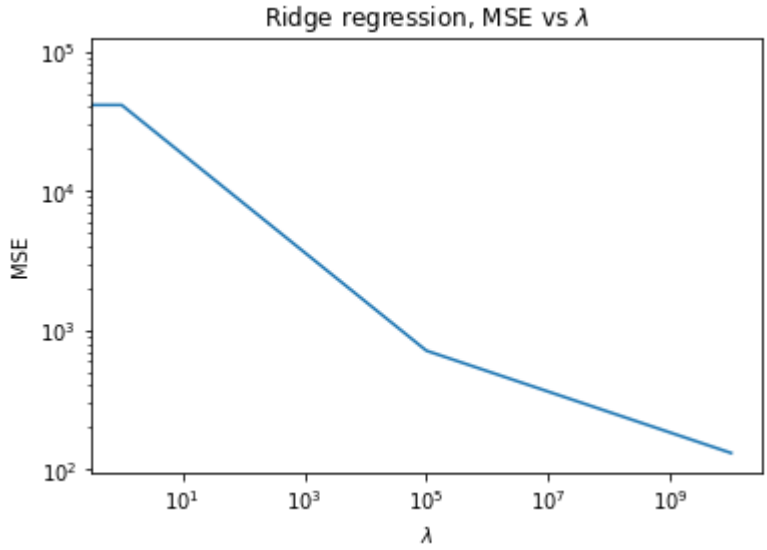
Now use "dataWeek4Test.npz" to validate the model for same values of λs ($\lambda = [0, 1, 10^5, 10^{10}]$) used in previous case. And plot λ vs validation error.

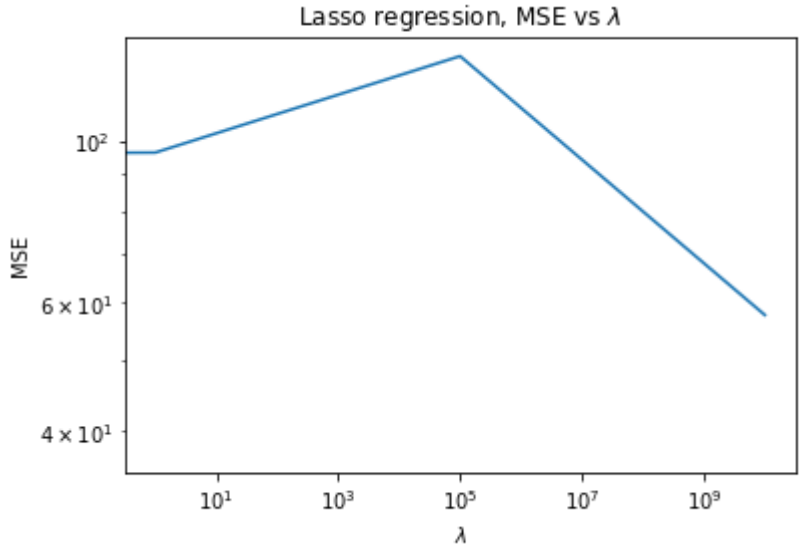
- Note : No need to plot the prediction on validation set since the validation points are very few
- Print the learned coefficients for Lasso and Ridge rigression and write your observations

Hint: Think of the property of Lasso Rigression (for writing observations)

In [65]:

```
# write your code here
X_test = data_test['X_test']
y_test = data_test['y_test']


X_test_reshaped = np.reshape(X_test,(X_test.size,1))
y_test_reshaped = np.reshape(y_test,(y_test.size,1))

poly = PolynomialFeatures(degree=9, include_bias=False)
X_test_poly = poly.fit_transform(X_test_reshaped)

#Training error vs lambda:
#for ridge:
mse_history=[]
for lamda in lamdas:
    Y_hat, mse , reg = regression(X_train_reshaped, y_train_reshaped, reg_type ='ridge'
, degree = 9, alpha=lamda)

    Y_test_hat = reg.predict(X_test_poly)
    mse = mean_squared_error(y_test_reshaped,Y_test_hat)
    mse_history.append(mse)

plt.loglog(lamdas,mse_history)

plt.xlabel(r'$\lambda $')
plt.ylabel('MSE')
plt.title('Ridge regression, Test MSE vs ' + r'$\lambda $')
plt.show();

#for lasso:
mse_history=[]
for lamda in lamdas:
    Y_hat, mse , reg = regression(X_train_reshaped, y_train_reshaped, reg_type ='lasso'
, degree = 9, alpha=lamda)
    Y_test_hat = reg.predict(X_test_poly)
    mse = mean_squared_error(y_test_reshaped,Y_test_hat)
    mse_history.append(mse)

plt.loglog(lamdas,mse_history)

plt.xlabel(r'$\lambda $')
plt.ylabel('MSE')
plt.title('Lasso regression, Test MSE vs ' + r'$\lambda $')
plt.show();
```





Questions and Observations

- What was your observation in simple linear rigression without regulaization when polynomial features of degree of 9 is used and what is the reason?

Ans: In simple linear rigression without regulaization when polynomial features of degree of 9 is used, the model overfits the data and hence does not generalise well on new data. Because of this overfitting, we use regularization to get a model which generalises well and does not overfit.

- As value of lambda increases what happens to the model complexity?

Ans: Model complexity decreases.

- What can be infered from the mean squared error versus lambda?

Ans: For training MSE, as lambda increases, MSE increases. As we are diminishing more and more coefficients. For Test MSE, an appropriate lambda gives low MSE i.e., the model is neither overfitted nor underfitted.

- Statement: Regularization gives finer control over fitting the data than using just change of degree of the polynomial features (True or False) Justify (not mathematical)

Ans: True. Regularized regression can have features of higher order but will diminish the effect of not so important features. This provides finer control over the fit.

- Comment on the train error vs λ and validation error vs λ what are the observations?

Ans: For lower lambdas, train error is very small but validation error is very high, due to overfitting. For very high lambdas, both the errors are high, due to underfitting. For an appropriate lambda, the model is neither overfitted nor underfitted and generalises well.

In [ ]: