1- option 2.

The test data is assumed to be sampled from the same unknown distribution that the training data is generated ($\mathcal{D}$). More information on Page 34, of "Understanding Machine Learning: From Theory to Algorithms" book.

2- option 1.

The probability of having more than of 10 noisy examples in a sample of size 35 drawn from a distribution with 20% noisy examples can be computed as:
$p(x > 10) = 1 - p(x \leq 10) = 1 - (p(x = 0) + \cdots + p(x = 10))$ where each of these probabilities is a binomial trail. Hence, the solution can be computed as: $1 - (\sum_{i=0}^{10} \binom{35}{i}(0.2)^i(0.8)^{35-i}) = 0.07$

In [1]:

```
import numpy as np
import math
def binom(n, k):
    #computes the binomial coefficient
    return math.factorial(n) // math.factorial(k) // math.factorial(n - k)
n=35
p=0.2
sum_prob=0
#sums up the probability of the cases where we have 10 or less noisy example
for N in range(11):
    sum_prob+=(binom(n, N)*np.power(p,N)*np.power(1-p,n-N))
print("probability of sampling 10 or less noisy examples:",np.round(sum_prob,2))
print("probability of sampling  more than 10 noisy examples: ",np.round(1-sum_pr
ob,2))
```

```
probability of sampling 10 or less noisy examples: 0.93
probability of sampling  more than 10 noisy examples:  0.07
```

3- option 2.

Note that in both cases, we want to increase TP cases.

A) In this problem, we are interested in decreasing FN (a term in the recall denominator), which corresponds to the cancer patients classified as healthy.

B) The goal is to avoid missing any non-spam email. Therefore reducing FP (a term in the precision denominator) will be ideal.

4- option 1

In [3]:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
#import the data
X = pd.read_csv("./X.csv")
Y = pd.read_csv("./Y.csv")
#splits the data to training and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, rando
m_state=1)
#trains the LR model
lin_model = LinearRegression(fit_intercept = False).fit(X_train, Y_train)
#prediction
y_test_predict = lin_model.predict(X_test)
#RMSE computation
rmse = (np.sqrt(mean_squared_error(Y_test, y_test_predict)))
print("root mean square error: ",round(rmse,2))
```

```
root mean square error:  5.26
```

Or you can simply follow the formulations in the lecture 1 to build the LR model as:

In [5]:

```
from numpy.linalg import inv
#use training set to compute the regression coefficient
X_arr=np.asanyarray(X_train)
Y_arr=np.asanyarray(Y_train)
#compute the regression coeffient beta, formula page 18, lecture 1
XTX=inv(np.matmul(np.transpose(X_arr),X_arr))
XTY=np.matmul(np.transpose(X_arr),Y_arr)
beta=np.matmul(XTX,XTY)
#prediction
y_pred=np.matmul(np.transpose(beta),np.transpose((np.asanyarray(X_test))))
#RMSE computation
rmse = (np.sqrt(mean_squared_error(Y_test, np.transpose(y_pred))))
print("root mean square error: ",round(rmse,2))
```

```
root mean square error:  5.26
```

5- option 3.

In [12]:

```
from sklearn.model_selection import KFold
def avg_var_CV(k):
    #computes mean and variance of the RMSEs over test folds
    #k in the number of folds for cross-validation
    rmse_list = list()
    #splits the training set to training and test folds
    kf = KFold(n_splits=k, random_state=1, shuffle=True)
    for train_index, test_index in kf.split(X_arr):
        xcv_train, xcv_test = X_arr[train_index], X_arr[test_index]
        ycv_train, ycv_test = Y_arr[train_index], Y_arr[test_index]
        #fits the LR model on trainin fold
        lin_model = LinearRegression(fit_intercept = False).fit(xcv_train, ycv_t
rain)
        #tests the model on test fold and computes RMSE
        rmse_list.append((np.sqrt(mean_squared_error(ycv_test, lin_model.predict
(xcv_test)))))
        # print the mean and variance of the RMSEs for the given k
    return print("K =",k,", ", "Average: ",round(np.mean(rmse_list),2),", ","Var
iance: ",round(np.var(rmse_list),2) )
avg_var_CV(2)
avg_var_CV(5)
```

```
K = 2 ,  Average:  6.03 ,  Variance:  0.37
K = 5 ,  Average:  5.72 ,  Variance:  0.5
```