Homework 2

Yanrong Wu 1801212952

Problem 1

1.Closed form function

Q: Implement a function closed_form_1 that computes this closed form solution given the features $\mathbf{X}$, labels Y (using Python or Matlab).

In [27]:
```
import numpy as np
import pandas as pd
from numpy.linalg import inv
import matplotlib.pyplot as plt

climate_change_1 = pd.read_csv('climate_change_1.csv')
climate_change_1.head()
```

Out[27]:

	Year	Month	MEI	CO2	CH4	N2O	CFC-11	CFC-12	TSI	Aerosols	Temp
0	1983	5	2.556	345.96	1638.59	303.677	191.324	350.113	1366.1024	0.0863	0.109
1	1983	6	2.167	345.52	1633.71	303.746	192.057	351.848	1366.1208	0.0794	0.118
2	1983	7	1.741	344.15	1633.22	303.795	192.818	353.725	1366.2850	0.0731	0.137
3	1983	8	1.130	342.25	1631.35	303.839	193.602	355.633	1366.4202	0.0673	0.176
4	1983	9	0.428	340.17	1648.40	303.901	194.392	357.465	1366.2335	0.0619	0.149

In [28]:
```
climate_change_1_train=climate_change_1.iloc[0:284]
#climate_change_1_train
climate_change_1_test=climate_change_1.iloc[284:308]
#climate_change_1_test
```

In [29]:
```
def closed_form_1(x,y):
    return np.linalg.inv(x.T @ x) @ (x.T @ y)

x_train= climate_change_1_train[['MEI','CO2','CH4','N2O','CFC-11','CFC-12','TSI',
x_train_withc = x_train.copy()
x_train_withc["constant"] = 1
x_train_withc = x_train_withc.values
y_train = climate_change_1_train["Temp"].values
theta0 = closed_form_1(x_train_withc,y_train)
theta0_pd = pd.DataFrame(theta1)
theta0_pd.columns=["Coefficient"]
theta0_pd.index=["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols","cons
theta0_pd
```

Out[29]:

	Coefficient
MEI	0.064205
CO2	0.006457
CH4	0.000124
N2O	-0.016528
CFC-11	-0.006630
CFC-12	0.003808
TSI	0.093141
Aerosols	-1.537613
constant	-124.594261

2.R2

Q: Write down the mathematical formula for the linear model and evaluate the model R2 on the training set and the testing set.

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots \dots + \beta_n x_{in} + \epsilon$$

where i is the index of samples

x_i is the expanatory variables, in this case, it means CO2, CH4, and other factors.

y_i is the dependent variables, in this case, it means Temperature.

β_0 is y intercept

β_i is coefficients for expanatory variables

ϵ is error term

In [30]:
```
y_hat = x_train_withc @ theta0
y_bar = np.sum(y_train) / len (y_train)

ESS = np.sum((y_hat -y_bar)**2)
TSS = np.sum((y_train - y_bar)**2)

r_square_test = ESS / TSS
r_square_test
```

Out[30]: 0.7508932744089848

In [31]:
```
x_test = climate_change_1_test[["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","
x_test_withc = x_test.copy()
x_test_withc["constant"] = 1
x_test_withc = x_test_withc.values
y_test = climate_change_1_test["Temp"].values
y_hat = x_test_withc @ theta0

y_bar = np.sum(y_test) / len (y_test)

ESS = np.sum((y_hat -y_bar)**2)
TSS = np.sum((y_test - y_bar)**2)

r_square_test = ESS / TSS
r_square_test
```

Out[31]: 0.22517701367302714

3.Significant Variables

Q: Which variables are significant in the model?

In [32]:
```
from scipy import stats
y_train_predict = x_train_withc @ theta0
MSE = (sum((y_train-y_train_predict)**2))/(len(x_train_withc)- np.shape(x_train_w
var_b = MSE * (np.linalg.inv(np.dot(x_train_withc.T,x_train_withc)).diagonal())
svar_b = np.sqrt(var_b)
ts_b = theta0/ svar_b
p_values =[2*(1-stats.t.cdf(np.abs(i),(len(x_train_withc)-1))) for i in ts_b]
result = pd.DataFrame()
result["P value"] = p_values
result["T statistic"] = ts_b
result.index=["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols","constan
result
```

Out[32]:

	P value	T statistic
MEI	0.000000e+00	9.923226
CO2	5.042596e-03	2.826420
CH4	8.101405e-01	0.240469
N2O	5.464021e-02	-1.929726
CFC-11	5.913566e-05	-4.077834
CFC-12	2.085760e-04	3.757293
TSI	1.057310e-09	6.312561
Aerosols	5.109246e-12	-7.210301
constant	1.381915e-09	-6.265174

MEI,CO2,CFC-11,CFC-12,TSI,Aerosols are significant in the model(0.05 significant level).

4. For climate_change_2.csv

Q: Write down the necessary conditions for using the closed form solution. And you can apply it to the dataset climate_change_2.csv, explain the solution is unreasonable.

The necessary conditions are:
X'X exists.

```
The expectation of error is 0.

The distribution of error obeys Normal dirstribution.
```

In [33]:
```
climate_change_2 = pd.read_csv('climate_change_2.csv')
#climate_change_2
```

In [34]:

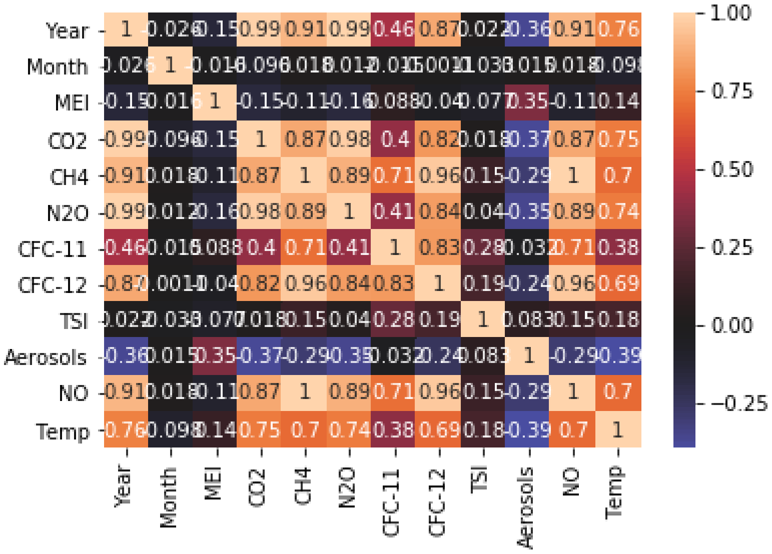
```python
climate_change_2_train=climate_change_2.iloc[0:284]
climate_change_2_test=climate_change_2.iloc[284:308]

def closed_form_1(x,y):
    return np.linalg.inv(x.T @ x) @ (x.T @ y)

x_train= climate_change_2_train[['MEI','CO2','CH4','N2O','CFC-11','CFC-12','TSI',
x_train_withc = x_train.copy()
x_train_withc["constant"] = 1
x_train_withc = x_train_withc.values
y_train = climate_change_2_train["Temp"].values
theta1 = closed_form_1(x_train_withc,y_train)
theta1_pd = pd.DataFrame(theta1)
theta1_pd.columns=["Coefficient"]
theta1_pd.index=["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols","cons
theta1_pd
#The solution is not reasonable.
```

Out[34]:

	Coefficient
MEI	0.064205
CO2	0.006457
CH4	0.000124
N2O	-0.016528
CFC-11	-0.006630
CFC-12	0.003808
TSI	0.093141
Aerosols	-1.537613
constant	-124.594261

In [35]:

```python
import pandas as pd
climate_change_2_corr = climate_change_2.corr()
# Visualization
import matplotlib.pyplot as mp, seaborn
seaborn.heatmap(climate_change_2_corr, center=0, annot=True)
mp.show()
```



It can be concluded from the correlation matrix that NO and CH4 are completely linearly correlated, so there is no inverse matrix, and the formula is invalid. So the solution is unreasonable.

Problem 2----Regularization

1.Loss Function

Q: Please write down the loss function for linear model with L1 regularization, L2 regularization, respectively.

In [36]:
```
#L1 regularization
def L1Norm(l, theta):
    return  np.dot(np.abs(theta), np.ones(theta.size)) * l

def L1NormPartial(l, theta):
    return np.sign(theta) * l

# For linear regression, the derivative of J function is:
def __Jfunction(self):
    sum = 0
    for i in range(0, self.m):
        err = self.__error_dist(self.x[i], self.y[i])
        sum += np.dot(err, err)
        sum += Regularization.L2Norm(0.8, self.theta)
        return 1/(2 * self.m) * sum
```

In [37]:
```
#L2 regularization
def L2Norm(l, theta):
    return  np.dot(theta, theta) * l

def L2NormPartial(l, theta):
    return theta * l

# For linear regression, the derivative of J function is:
def __partialderiv_J_func(self):
        sum = 0
        for i in range(0, self.m):
            err = self.__error_dist(self.x[i], self.y[i])
            sum += np.dot(self.x[i], err)
            sum += Regularization.L2NormPartial(0.8, self.theta)
            return 1/self.m * sum
```

Loss function for linear model with L1 regularization:

$$\frac{1}{2m}[\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{n}|\theta_j|]$$

where m is the number of samples, n is the number of explanatory variables

Loss function for linear model with L2 regularization:

$$\frac{1}{2m}[\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 + \lambda \sum_{j=1}^{n}\theta_j^2]$$

2.Closed Form Solution

Q: The closed form solution for linear model with L2 regularization: $\mathbf{\theta} = (\mathbf{XTX} + \lambda\mathbf{I}){-}\mathbf{1XTY}$ where I is the identity matrix. Write a function closed_form_2 that computes this closed form solution given the features X, labels Y and the regularization parameter λ.

We can answer questions 2 and 4 together.

In [38]:
```
def closed_form_2(x,y,lambda1):
    I_m = np.identity(np.shape(x)[1])
    I_m[np.shape(x)[1] - 1,np.shape(x)[1] - 1] = 0
    return np.linalg.inv(x.T @ x + lambda1 * I_m) @ (x.T @ y)
```

In [39]:
```
theta2 = closed_form_2(x_train_withc, y_train, 1)
theta2_pd = pd.DataFrame(theta2)
theta2_pd.columns=["Coefficient with L2"]
theta2_pd.index=["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols","cons
theta2_pd
```

Out[39]:

	Coefficient with L2
MEI	0.049887
CO2	0.008444
CH4	0.000277
N2O	-0.020004
CFC-11	-0.007224
CFC-12	0.003970
TSI	0.077529
Aerosols	-0.238930
constant	-103.116523

3.Comparasion

Q: Compare the two solutions in problem 1 and problem 2 and explain the reason why linear model with L2 regularization is robust. (using climate_change_1.csv)

In [40]:
```
theta2_pd_compare = theta2_pd.copy()
theta2_pd_compare["Coefficient without L2"] = theta1
theta2_pd_compare
```

Out[40]:

	Coefficient with L2	Coefficient without L2
MEI	0.049887	0.064205
CO2	0.008444	0.006457
CH4	0.000277	0.000124
N2O	-0.020004	-0.016528
CFC-11	-0.007224	-0.006630
CFC-12	0.003970	0.003808
TSI	0.077529	0.093141
Aerosols	-0.238930	-1.537613
constant	-103.116523	-124.594261

Actually,without L2, the R squre is lower than that with L2. It will reduce the coefficient of unimportant prediction factors close to 0 and avoid overfitting. In L2 model, it is less sensitive to single variable, so it is more robust.

4.Change the regularization parameter λ

Q: You can change the regularization parameter λ to get different solutions for this problem. Suppose we set λ = 10, 1, 0.1, 0.01, 0.001, and please evaluate the model R2 on the training set and the testing set. Finally, please decide the best regularization parameter λ. (Note that: As a qualified data analyst, you must know how to choose model parameters, please learn about cross validation methods.)

In [41]:
```
def closed_form_2():

    dataset = pd.read_csv("climate_change_1.csv")
    X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols"])

    y = dataset.get("Temp")

    X = np.column_stack((X,np.ones(len(X))))

    for lambda1 in [10,1,0.1,0.01,0.001]:
        X_train = X[:284]
        X_test = X[284:]
        y_train = y[:284]
        y_test = y[284:]

        X_train=np.mat(X_train)
        y_train = np.mat(y_train).T
        xTx = X_train.T*X_train
        w = 0
        print("="*25+"L2 Regularization (lambda is "+str(lambda1)+")"+"="*25)
        I_m= np.eye(X_train.shape[1])
        if np.linalg.det(xTx+lambda1*I_m)==0.0:
            print("xTx is invertible")
        else:
            print(np.linalg.det(xTx+lambda1*I_m))
            w= (xTx+lambda1*I_m).I*(X_train.T*y_train)
        wights = np.ravel(w)
        y_train_pred = np.ravel(np.mat(X_train)*np.mat(w))
        y_test_pred = np.ravel(np.mat(X_test)*np.mat(w))
        coef_=wights[:-1]
        intercept_=wights[-1]

        X_train = np.ravel(X_train).reshape(-1,9)
        y_train = np.ravel(y_train)

        print("Coefficient: ",coef_)
        print("Intercept: ",intercept_)
        print("the model is: y = ",coef_,"* X +(",intercept_,")")
        y_train_avg = np.average(y_train)

        R2_train = 1-(np.average((y_train-y_train_pred)**2))/(np.average((y_train
        print("R2 in Train : ",R2_train)

        y_test_avg = np.average(y_test)
        R2_test = 1-(np.average((y_test-y_test_pred)**2))/(np.average((y_test-y_t
        print("R2 in Test : ",R2_test)

closed_form_2()
#This part I have discussed with my classmate and searched a lot on the website.
```

```
=========================L2 Regularization (lambda is 10)======================
===
4.052005289688253e+33
Coefficient:  [ 0.04054315  0.00814554  0.00020508 -0.01608137 -0.00636145  0.0
03689
  0.00126458 -0.02443305]
Intercept:  -0.0002202205828863274
the model is:  y =  [ 0.04054315  0.00814554  0.00020508 -0.01608137 -0.0063614
5  0.003689
  0.00126458 -0.02443305] * X +( -0.0002202205828863274 )
R2 in Train :   0.6803719394071281
R2 in Test :   -0.7061640575416965

=========================L2 Regularization (lambda is 1)=======================
```

```
L2 Regularization (lambda is 1)
==
4.182558175861993e+31
Coefficient:  [ 0.04395558  0.00804313  0.00021395 -0.01693027 -0.00646627  0.0
0376881
  0.00146759 -0.21177258]
Intercept:  -0.0022945422838525635
the model is:  y =  [ 0.04395558  0.00804313  0.00021395 -0.01693027 -0.0064662
7  0.00376881
  0.00146759 -0.21177258] * X +( -0.0022945422838525635 )
R2 in Train :   0.6897571586198687
R2 in Test :   -0.5861726468586046
========================L2 Regularization (lambda is 0.1)=====================
====
1.0051083854786037e+30
Coefficient:  [ 5.06851277e-02  6.98925378e-03  1.30761990e-04 -1.48156599e-02
 -6.07864608e-03  3.66100278e-03  1.36118274e-03 -8.71332452e-01]
Intercept:  -0.025045661913281534
the model is:  y =  [ 5.06851277e-02  6.98925378e-03  1.30761990e-04 -1.4815659
9e-02
 -6.07864608e-03  3.66100278e-03  1.36118274e-03 -8.71332452e-01] * X +( -0.025
045661913281534 )
R2 in Train :   0.7110310866063567
R2 in Test :   -0.36213522139292387
========================L2 Regularization (lambda is 0.01)====================
=====
6.930175866500259e+28
Coefficient:  [ 5.46344723e-02  6.35012916e-03  7.94610956e-05 -1.34794077e-02
 -5.83699154e-03  3.59093203e-03  1.44947810e-03 -1.26505174e+00]
Intercept:  -0.26232414556713424
the model is:  y =  [ 5.46344723e-02  6.35012916e-03  7.94610956e-05 -1.3479407
7e-02
 -5.83699154e-03  3.59093203e-03  1.44947810e-03 -1.26505174e+00] * X +( -0.262
32414556713424 )
R2 in Train :   0.7153953027375966
R2 in Test :   -0.24446585990645597
========================L2 Regularization (lambda is 0.001)===================
======
6.742979655964518e+27
Coefficient:  [ 5.53981612e-02  6.25686043e-03  7.26293229e-05 -1.33359358e-02
 -5.81554289e-03  3.58444627e-03  3.15485922e-03 -1.32868779e+00]
Intercept:  -2.5924696366355677
the model is:  y =  [ 5.53981612e-02  6.25686043e-03  7.26293229e-05 -1.3335935
8e-02
 -5.81554289e-03  3.58444627e-03  3.15485922e-03 -1.32868779e+00] * X +( -2.592
4696366355677 )
R2 in Train :   0.7168000467674187
R2 in Test :   -0.21604536980238898
```

The bast lambda is 0.001 in 0.05 significant level.

Problem 3 — Feature Selection

1.Workflow

Q: From Problem 1, you can know which variables are significant, therefore you can use less variables to train model. For example, remove highly correlated and redundant features. You can propose a workflow to select feature.

☐ Find the highly correlated features

☐ Calculate the vif of these features to do feature selection

☐ Drop the features which are highly correlated with others and not important features

2.Better Model

Train a better model than the model in Problem 2.

In [42]:
```
dataset = pd.read_csv("climate_change_1.csv")
X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols"])
X_train = X[:284]
correlation = X_train.corr()
correlation
```

Out[42]:

	MEI	CO2	CH4	N2O	CFC-11	CFC-12	TSI	Aerosols
MEI	1.000000	-0.041147	-0.033419	-0.050820	0.069000	0.008286	-0.154492	0.340238
CO2	-0.041147	1.000000	0.877280	0.976720	0.514060	0.852690	0.177429	-0.356155
CH4	-0.033419	0.877280	1.000000	0.899839	0.779904	0.963616	0.245528	-0.267809
N2O	-0.050820	0.976720	0.899839	1.000000	0.522477	0.867931	0.199757	-0.337055
CFC-11	0.069000	0.514060	0.779904	0.522477	1.000000	0.868985	0.272046	-0.043921
CFC-12	0.008286	0.852690	0.963616	0.867931	0.868985	1.000000	0.255303	-0.225131
TSI	-0.154492	0.177429	0.245528	0.199757	0.272046	0.255303	1.000000	0.052117
Aerosols	0.340238	-0.356155	-0.267809	-0.337055	-0.043921	-0.225131	0.052117	1.000000

In [43]:
```
import numpy as np
import pandas as pd
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn import linear_model

#Variance Inflation Factor
def vif(X, thres=10.0):
    col = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:,col].values, ix) for ix in rang
        maxvif = max(vif)
        maxix = vif.index(maxvif)
        if maxvif > thres:
            del col[maxix]
            print('delete=',X.columns[col[maxix]],'  ', 'vif=',maxvif )
            dropped = True
    print('Remain Variables:', list(X.columns[col]))
    print('VIF:', vif)
    return list(X.columns[col])

dataset = pd.read_csv("climate_change_1.csv")
X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols"])

y = dataset.get("Temp")

X_train = X[:284]
X_test = X[284:]
y_train = y[:284]
y_test = y[284:]
d = vif(X_train)
print(d)

X = dataset.get( ['MEI', 'CFC-12', 'Aerosols'])
y = dataset.get("Temp")
X_train = X[:284]
X_test = X[284:]
y_train = y[:284]
y_test = y[284:]

regr = linear_model.LinearRegression()
regr.fit(X_train,y_train)
print('coefficients(b1,b2...):',regr.coef_)
print('intercept(b0):',regr.intercept_)
y_train_pred = regr.predict(X_train)

R2_1 = regr.score(X_train, y_train)
print(R2_1)
R2_2 = regr.score(X_test, y_test)
print(R2_2)
```

```
delete= CFC-11    vif= 239743.2424704495
delete= Aerosols    vif= 29867.18540477364
delete= CFC-11    vif= 11884.79599294173
delete= CFC-12    vif= 502.06957361985695
delete= CFC-12    vif= 122.31236225671839
Remain Variables: ['MEI', 'CFC-12', 'Aerosols']
VIF: [1.288871669460935, 1.33868239281389, 1.48103752609454]
['MEI', 'CFC-12', 'Aerosols']
coefficients(b1,b2...): [ 5.54993375e-02  1.86365387e-03 -2.08242114e+00]
intercept(b0): -0.6553255026654846
```

```
0.5996443150479794
0.004717686204946725
```

Problem 4 — Gradient Descent

Gradient descent algorithm is an iterative process that takes us to the minimum of a function. Please write down the iterative expression for updating the solution of linear model and implement it using Python or Matlab in gradientDescent function.

P.S.: The Gradient Descent fomula can see from the pic.Gradient Descent in the folder. (Because there are something wrong so I save the fomula in the picture.)

In [ ]:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def costFunc(X,Y,theta):
    #cost func
    inner=np.power((X*theta.T)-Y,2)
    return np.sum(inner)/(2*len(X))

def gradientDescent(X,Y,theta,alpha,iters):
    temp = np.mat(np.zeros(theta.shape))
    cost = np.zeros(iters)
    thetaNums = int(theta.shape[1])

    for i in range(iters):
        error = (X*theta.T-Y)
        for j in range(thetaNums):
            derivativeInner = np.multiply(error,X[:,j])
            temp[0,j] = theta[0,j]-(alpha*np.sum(derivativeInner)/len(X))
        theta = temp
        cost[i]=costFunc(X,Y,theta)
    return theta,cost


dataset = pd.read_csv("climate_change_1.csv")
X = dataset.get(["MEI","CO2","CH4","N2O","CFC-11","CFC-12","TSI","Aerosols"])

y = dataset.get("Temp")
X = np.column_stack((np.ones(len(X)),X))
X_train = X[:284]
X_test = X[284:]
y_train = y[:284]
y_test = y[284:]

X_train = np.mat(X_train)
Y_train = np.mat(y_train).T

for i in range(1,9):
    X_train[:,i] = (X_train[:,i] - min(X_train[:,i])) / (max(X_train[:,i]) - min(

theta_n = (X_train.T*X_train).I*X_train.T*Y_train
print("theta =",theta_n)
theta = np.mat([0,0,0,0,0,0,0,0,0])
iters = 100000
alpha = 0.001

finalTheta,cost = gradientDescent(X_train,Y_train,theta,alpha,iters)
print("final theta ",finalTheta)
print("cost ",cost)

fig, bx = plt.subplots(figsize=(8,6))
bx.plot(np.arange(iters), cost, 'r')
bx.set_xlabel('Iterations')
bx.set_ylabel('Cost')
bx.set_title('Error vs. Training Epoch')
plt.show()
```

```
theta = [[-0.07698894]
 [ 0.29450977]
 [ 0.28935427]
 [ 0.02211171]
 [-0.27724073]
 [ 0.53156629]
```

```
[-0.53136629]
[ 0.7376296 ]
[ 0.17604596]
[-0.22725924]]
```