Conceptual problems:

```
import numpy as np
import math
from sklearn.model_selection import train_test_split
import pandas as pd
import random
import matplotlib.pyplot as plt
```

QC.1

```
df_data = pd.read_csv('gss_train.csv')
```

```
X = np.array(df_data.iloc[:,:20])[:1000]
```

```
random.seed(2020)
B = np.random.random(20)
B[-5:] = 0
```

```
random.seed(2020)
error = np.array([np.random.normal(loc=0.0, scale=0.5, size=None) for i in
range(1000)])
Y = np.dot(X,B) + error
```

QC.2

```
x_train, x_test, y_train, y_test = \
train_test_split(X,Y,test_size = 0.9,shuffle = True)
print(f'train set length:{len(x_train)}')
print(f'test set length:{len(x_test)}')
```

```
train set length:100
test set length:900
```

```
from tqdm import tnrange, tqdm_notebook
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import itertools
from collections import defaultdict
```

```
MSE_dict = {}
size_to_mse = defaultdict(list)
for k in range(1, 21):
    for combo in itertools.combinations(list(range(20)),k):
        model_k = linear_model.\
        LinearRegression(fit_intercept = 
True).fit(x_train[:,np.array(combo)],y_train)
        MSE_dict[combo] = mean_squared_error\
        (y_train,model_k.predict(x_train[:,np.array(combo)]))
        size_to_mse[len(combo)].append(MSE_dict[combo])
print('training period')

size_to_minmse = {length: min(size_to_mse[length]) for length in size_to_mse}
```

```
training period
```

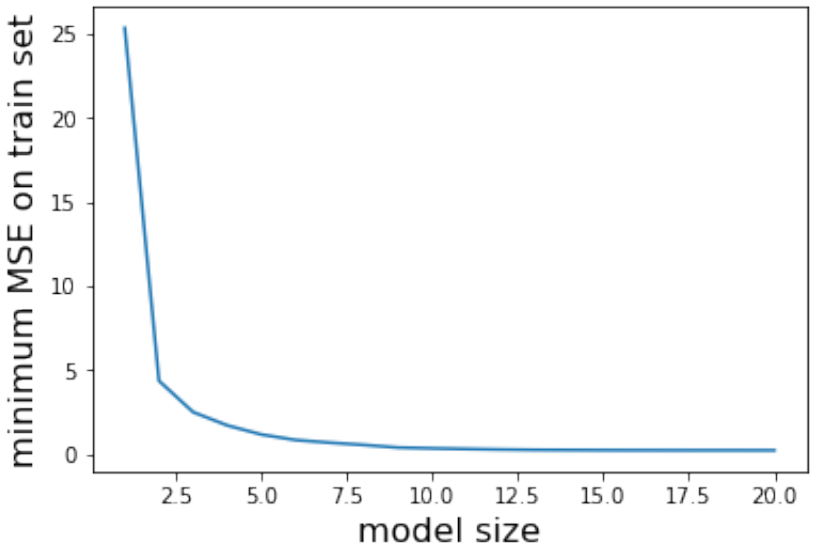
```
size_to_minmse# training MSE on best models of different sizes
```

```
{1: 25.329728017494155,
 2: 4.3405131195987643,
 3: 2.47985802683373,
 4: 1.6884604126319089,
 5: 1.1451976537073825,
 6: 0.81945605867775162,
 7: 0.66180280818065695,
 8: 0.52694081836476003,
 9: 0.37064483145236038,
 10: 0.32515906295810632,
 11: 0.28766125408522342,
 12: 0.25641050516468605,
 13: 0.23370102958039154,
 14: 0.22512011173336952,
 15: 0.21551475938414644,
 16: 0.21186980680938672,
 17: 0.20886263575569064,
 18: 0.20762603916030409,
 19: 0.20699969831020568,
```

```
20: 0.20671365775352726}
```

QC.3

```
plt.plot(list(map(int,size_to_minmse.keys())),size_to_minmse.values())
plt.xlabel('model size',size = 16)
plt.ylabel('minimum MSE on train set',size = 16)
plt.title('MSE on different model size')
plt.show()
```



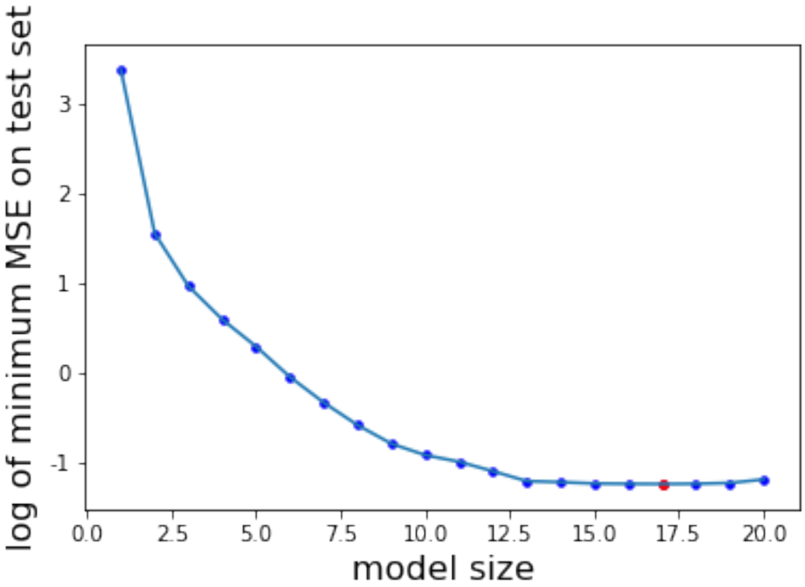
When model's p = 20, we get the minimum MSE on training set.

```
MSE_dict_test = {}
size_to_mse_test = defaultdict(list)
for k in range(1, 21):
    for combo in itertools.combinations(list(range(20)),k):
        model_k = linear_model.LinearRegression(fit_intercept = True).\
        fit(x_train[:,np.array(combo)],y_train)

        MSE_dict_test[combo] = mean_squared_error\
        (y_test,model_k.predict(x_test[:,np.array(combo)]))
        size_to_mse_test[len(combo)].append(MSE_dict_test[combo])
print('training period')

size_to_minmse_test = {length: min(size_to_mse_test[length]) for length in
size_to_mse_test}
```

```
training period
```

QC.4

```
plt.plot(list(map(int,size_to_minmse_test.keys())),\
         np.log(np.array(list(map(float,size_to_minmse_test.values())))))
plt.xlabel('model size',size = 16)
plt.ylabel('log of minimum MSE on test set',size = 16)
plt.scatter(size_to_minmse_test.keys(),\

 np.log(np.array(list(map(float,size_to_minmse_test.values())))),c='b',s = 12)
plt.scatter(17,math.log(size_to_minmse_test[17]),c = 'r',s = 15)
plt.show()
```




```
size_to_minmse_test
```

```
{1: 29.350384110017782,
 2: 4.7327205679447895,
 3: 2.6261884057769977,
 4: 1.8145695156262516,
 5: 1.3415291590057667,
 6: 0.95651530643531235,
 7: 0.71914872543995145,
 8: 0.5595516746309569,
 9: 0.45500109460487376,
 10: 0.40077050223455835,
 11: 0.37223355223065935,
 12: 0.33496846189463375,
 13: 0.29990368037199577,
```

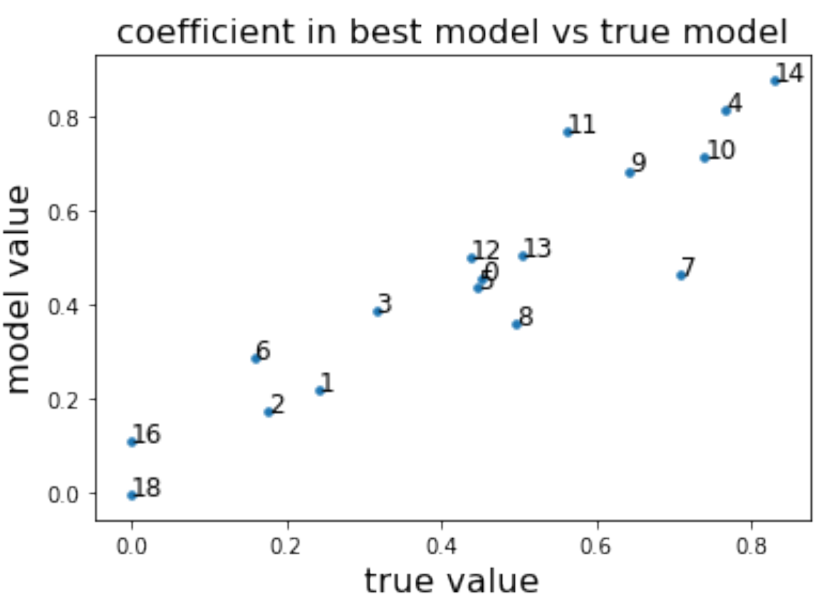
```
14: 0.29667365202451784,
15: 0.29214762740211592,
16: 0.29110624297660642,
17: 0.29052866061394222,
18: 0.29123606149243653,
19: 0.29387748022116389,
20: 0.30553473644177692}
```

QC.5

When there are 17 coefficients in our best model, the test MSE is minimized. This is due to the fact that model of 20 variables already overfit the noise in our dataset, and give a relatively poorer performance on test set than the best model.

```
mse_test_reverse = {MSE_dict_test[combo]:combo for combo in MSE_dict_test}
best_combo_17 = sorted(mse_test_reverse.items(),key = lambda x:x[0])[0][1]
```

```
model_k = linear_model.LinearRegression(fit_intercept =
True).fit(x_train[:,np.array(best_combo_17)],y_train)
plt.scatter(B[np.array(best_combo_17)],model_k.coef_,s = 12)
for i in range(len(model_k.coef_)):
    plt.text(B[np.array(best_combo_17)][i], model_k.coef_[i],
str(best_combo_17[i]),size = 12)
plt.xlabel('true value',size = 16)
plt.ylabel('model value',size = 16)
plt.title('coefficient in best model vs true model',size = 16)
plt.show()
```



QC.6

we found that the coefficient generated by the best model with the minimum MSE on test set is proportional to that in true model. Moreover, the best model choose 17 coefficients, which is close to the fact that there are 5 coefficients in reality are equal to 0.

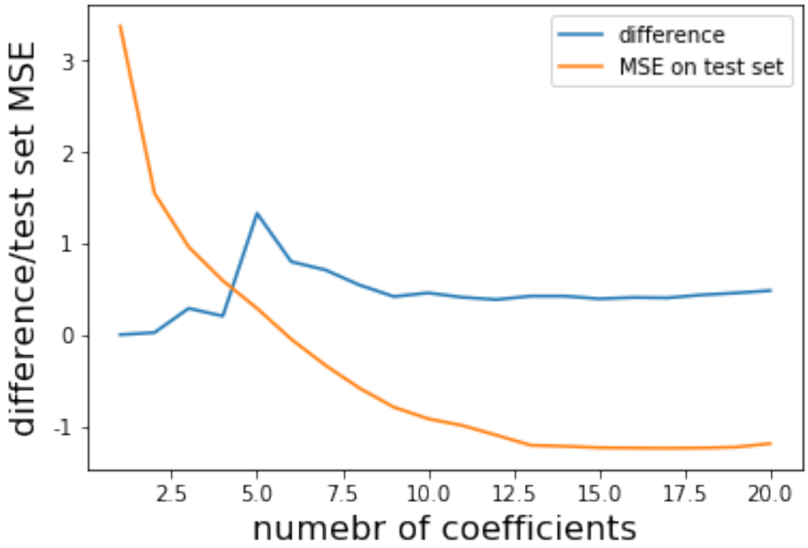
```
difference = []
for p in range(1,21):
    compare_dict = {}
    for combo in MSE_dict_test:
        if len(combo) == p:
            compare_dict[combo] = MSE_dict_test[combo]
    best_combo = sorted(compare_dict.items(),key=lambda x:x[1])[0][0]
    model_k = linear_model.LinearRegression(fit_intercept = 
True).fit(x_train[:,np.array(best_combo)],y_train)
    difference.append(math.sqrt(np.sum((B[np.array(best_combo)] - 
model_k.coef_)**2)))
```

```
np.sum(B[np.array(best_combo)] - model_k.coef_**p)
```

```
-21.522890074576551
```

QC.7

```
plt.plot(range(1,21),difference,label = 'difference')
plt.plot(list(map(int,size_to_minmse_test.keys())),\
         np.log(np.array(list(map(float,size_to_minmse_test.values())))),label=
'MSE on test set')
plt.xlabel('numebr of coefficients',size = 16)
plt.ylabel('difference/test set MSE',size = 16)
plt.legend()
plt.show()
```



As the number of coefficients in best models increases, the plot of difference goes up at first, and then decrease sharply. In fact, when the MSE on test set drops and becomes stable as the number of coefficients approach that in true model, the difference between the coefficients calculated by best models and that in true model is close to 0.

when the model use inappropriate numbers of variables to configure a model, although we may get a lower test mse as we increase the number of variables contained in that model, we may still get a different model compared with the 'ture model'. But as we approach the true variables number in true model, we can both get a low difference and a low test MSE.

Application: prediction on individual's egalitarianism

```
import copy
```

```
df_data_train = pd.read_csv('gss_train.csv')
df_data_test = pd.read_csv('gss_test.csv')
```

```
Y_train = copy.copy(df_data_train['egalit_scale'])
Y_test = copy.copy(df_data_test['egalit_scale'])
df_data_train.drop(['egalit_scale'],axis = 1, inplace=True)
df_data_test.drop(['egalit_scale'],axis = 1, inplace=True)
```

```
X_train = np.array(df_data_train)
X_test = np.array(df_data_test)
Y_train = np.array(Y_train)
Y_test = np.array(Y_test)
```

QA.1

linear model

```
model_k = linear_model.LinearRegression(fit_intercept =
True).fit(X_train,Y_train)
linear_mse = mean_squared_error(Y_test,model_k.predict(X_test))
```

```
linear_mse
```

```
63.213629623015009
```

QA.2

Ridge

```
from sklearn.linear_model import Ridge
```

```
from sklearn.model_selection import KFold
```

```
indice = np.array(list(range(len(X_train))))
np.random.shuffle(indice)
kf = KFold(n_splits=10)
MSE_list = []
for lambdaa in range(70,90):
    mse_list = []
    for train_indice, test_indice in kf.split(indice):
        #print(train_indice.shape,test_indice.shape)
        clf = Ridge(alpha=lambdaa)
        clf.fit(X_train[train_indice],Y_train[train_indice])

 mse_list.append(mean_squared_error(Y_train[test_indice],clf.predict(X_train[te
st_indice])))
    MSE_list.append(np.mean(mse_list))
    print(lambdaa,MSE_list[-1])
```

```
70 60.7404227884
71 60.7389457121
72 60.7376202379
73 60.7364427498
74 60.7354097408
75 60.7345178084
```

```
76 60.7337636505
77 60.7331440614
78 60.732655928
79 60.7322962265
80 60.7320620189
81 60.7319504498
82 60.7319587433
83 60.7320842001
84 60.732324195
85 60.7326761739
86 60.7331376515
87 60.733706209
88 60.7343794915
89 60.735155206
```

```
clf = Ridge(alpha=81)
clf.fit(X_train,Y_train)
mean_squared_error(Y_test,clf.predict(X_test))
```

```
62.182092321036123
```

QA.3

Lasso

```
from sklearn.linear_model import Lasso
```

```
indice = np.array(list(range(len(X_train))))
np.random.shuffle(indice)
kf = KFold(n_splits=10)
MSE_list = []
for lambdaa in list(np.linspace(0.05,1,20)):
    mse_list = []
    for train_indice, test_indice in kf.split(indice):
        #print(train_indice.shape,test_indice.shape)
        clf = Lasso(alpha=lambdaa)
        clf.fit(X_train[train_indice],Y_train[train_indice])

 mse_list.append(mean_squared_error(Y_train[test_indice],clf.predict(X_train[te
st_indice])))
    MSE_list.append(np.mean(mse_list))
    print(lambdaa,MSE_list[-1])
```

```
0.05 59.8143249225
0.1 59.2832577093
0.15 59.5844409401
0.2 60.2216199025
0.25 60.9876981583
0.3 61.5551977997
0.35 62.1600762833
0.4 62.6545165477
0.45 63.0542228777
0.5 63.3729520983
0.55 63.6822465084
0.6 63.9943469292
0.65 64.331351907
0.7 64.6948161594
0.75 65.069231629
0.8 65.4661046849
0.85 65.8809048176
0.9 66.3179481501
0.95 66.7824084717
1.0 67.2715303408
```

```
# test set performance
clf = Lasso(alpha=0.1)
clf.fit(X_train,Y_train)
mean_squared_error(Y_test,clf.predict(X_test))
```

```
62.778415554773893
```

```
# coefficients larger than 0
df_data_train.columns[clf.coef_>0],clf.coef_[clf.coef_>0]
```

```
(Index(['black', 'childs', 'happy', 'owngun', 'pray', 'sex', 'sibs', 'tvhours',
        'spend3_Liberal'],
       dtype='object'),
 array([ 0.29850747,  0.33289112,  0.33829207,  0.61368689,  0.05670716,
         0.92432543,  0.15633715,  0.26472958,  1.17485402]))
```

QA.4

Elastic Net

```
from sklearn.linear_model import ElasticNet
```

```
np.linspace(0.05,0.55,11)
```

```
array([ 0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ,  0.35,  0.4 ,  0.45,
        0.5 ,  0.55])
```

```
indice = np.array(list(range(len(X_train))))
np.random.shuffle(indice)
kf = KFold(n_splits=10)
MSE_list = dict()
ratio = 0.05
for ratio in list(np.linspace(0,1,11)):
    for lambdaa in list(np.linspace(0.05,0.55,11)):
        mse_list = []
        for train_indice, test_indice in kf.split(indice):
            #print(train_indice.shape,test_indice.shape)
            clf = ElasticNet(alpha=lambdaa,l1_ratio= ratio)
            clf.fit(X_train[train_indice],Y_train[train_indice])

 mse_list.append(mean_squared_error(Y_train[test_indice],clf.predict(X_train[te
st_indice])))
        MSE_list[(ratio,lambdaa)] = np.mean(mse_list)
```

```
        print(lambdaa,ratio, MSE_list[(ratio,lambdaa)])
```

```
/Users/apple/anaconda3/lib/python3.6/site-
packages/sklearn/linear_model/coordinate_descent.py:491: ConvergenceWarning:
Objective did not converge. You might want to increase the number of
iterations. Fitting data with very small alpha may cause precision problems.
  ConvergenceWarning)
```

```
0.05 0.0 60.7464748394
0.1 0.0 60.8451993979
0.15 0.0 61.1525302037
0.2 0.0 61.5118217248
0.25 0.0 61.8764845067
0.3 0.0 62.2309404763
0.35 0.0 62.5702981243
0.4 0.0 62.8936846367
0.45 0.0 63.2017546936
0.5 0.0 63.4956895661
0.55 0.0 63.7767828009
0.05 0.1 60.6613597146
0.1 0.1 60.6863492409
0.15 0.1 60.9553030462
0.2 0.1 61.3009275275
0.25 0.1 61.6612417587
0.3 0.1 62.0102566026
0.35 0.1 62.3600475034
0.4 0.1 62.7026692608
0.45 0.1 63.0322724506
0.5 0.1 63.3501117665
0.55 0.1 63.6543029179
0.05 0.2 60.5752845108
0.1 0.2 60.5225169043
0.15 0.2 60.7587370463
0.2 0.2 61.076832788
0.25 0.2 61.4151978557
0.3 0.2 61.7768588259
0.35 0.2 62.1557521417
0.4 0.2 62.5303254069
0.45 0.2 62.8912287826
0.5 0.2 63.2398750932
0.55 0.2 63.5750500473
0.05 0.3 60.4925768072
0.1 0.3 60.3601310137
0.15 0.3 60.5476646937
0.2 0.3 60.8266205106
0.25 0.3 61.1930563736
0.3 0.3 61.6012611518
0.35 0.3 62.0175353855
```

```
0.4 0.3 62.4352462494
0.45 0.3 62.8430162068
0.5 0.3 63.2398736164
0.55 0.3 63.6242615294
0.05 0.4 60.4079522942
0.1 0.4 60.1837060158
0.15 0.4 60.3134472406
0.2 0.4 60.6144417384
0.25 0.4 61.0316424967
0.3 0.4 61.4928819469
0.35 0.4 61.9681459865
0.4 0.4 62.4403888143
0.45 0.4 62.8998790496
0.5 0.4 63.3323388951
0.55 0.4 63.7476504579
0.05 0.5 60.313573911
0.1 0.5 60.0052223352
0.15 0.5 60.0968185183
0.2 0.5 60.456253462
0.25 0.5 60.9305915837
0.3 0.5 61.4538639315
0.35 0.5 61.9891605858
0.4 0.5 62.5125061698
0.45 0.5 62.9971912653
0.5 0.5 63.4731205618
0.55 0.5 63.9397421721
0.05 0.6 60.2065848425
0.1 0.6 59.8089394171
0.15 0.6 59.938325728
0.2 0.6 60.3363899674
0.25 0.6 60.8693664128
0.3 0.6 61.4571699978
0.35 0.6 62.0498323278
0.4 0.6 62.5998836481
0.45 0.6 63.1389073539
0.5 0.6 63.6480874298
0.55 0.6 64.1220498265
0.05 0.7 60.0985921075
0.1 0.7 59.6428180158
0.15 0.7 59.782671897
0.2 0.7 60.2429367629
0.25 0.7 60.8437273498
0.3 0.7 61.4963759415
0.35 0.7 62.1131077272
0.4 0.7 62.6897036716
0.45 0.7 63.225938882
0.5 0.7 63.7453872903
0.55 0.7 64.2486953912
0.05 0.8 59.9812787509
```

```
0.1 0.8 59.5031858576
0.15 0.8 59.6701245176
0.2 0.8 60.1897987384
0.25 0.8 60.8505275734
0.3 0.8 61.5426037616
0.35 0.8 62.1384443774
0.4 0.8 62.7085916379
0.45 0.8 63.2826086766
0.5 0.8 63.8035335361
0.55 0.8 64.2535333405
0.05 0.9 59.8870563492
0.1 0.9 59.3873865141
0.15 0.9 59.6023322836
0.2 0.9 60.1760278531
0.25 0.9 60.8918537463
0.3 0.9 61.5438312808
0.35 0.9 62.1284094407
0.4 0.9 62.7224612499
0.45 0.9 63.2321189863
0.5 0.9 63.6643161664
0.55 0.9 64.044619933
0.05 1.0 59.8143249225
0.1 1.0 59.2832577093
0.15 1.0 59.5844409401
0.2 1.0 60.2216199025
0.25 1.0 60.9876981583
0.3 1.0 61.5551977997
0.35 1.0 62.1600762833
0.4 1.0 62.6545165477
0.45 1.0 63.0542228777
0.5 1.0 63.3729520983
0.55 1.0 63.6822465084
```

```
# the best combination
sorted(MSE_list.items(),key = lambda x:x[1])[0]# (alpha = 1,lambda = 0.1)
```

```
((1.0, 0.10000000000000001), 59.283257709253562)
```

```
# test set performance
clf = ElasticNet(alpha=0.1,l1_ratio=1)
clf.fit(X_train,Y_train)
mean_squared_error(Y_test,clf.predict(X_test))
```

```
62.778415554773893
```

```
# coefficients larger than 0
df_data_train.columns[clf.coef_>0],clf.coef_[clf.coef_>0]
```

```
(Index(['black', 'childs', 'happy', 'owngun', 'pray', 'sex', 'sibs', 'tvhours',
        'spend3_Liberal'],
       dtype='object'),
 array([ 0.29850747,  0.33289112,  0.33829207,  0.61368689,  0.05670716,
         0.92432543,  0.15633715,  0.26472958,  1.17485402]))
```

QA.5

In conclusion, we can predict a person's egalitarianism with the mse of about 62, and when there are many variables, linear regression gives a poor performance while regularization helps to improve the performance. Among the regularization models we configured there(ridge, lasso, elastic net), there is no distinct difference.