```
import numpy as np
import pandas as pd
import copy
import math
import matplotlib.pyplot as plt
```

```
df_train = pd.read_csv('gss_train.csv')
df_test = pd.read_csv('gss_test.csv')
```

```
df_train['egalit_scale'].describe()
```

```
count    1481.000000
mean       19.430115
std         9.621450
min         1.000000
25%        13.000000
50%        20.000000
75%        27.000000
max        35.000000
Name: egalit_scale, dtype: float64
```

```
df_train['income06'].describe()
```

```
count    1481.000000
mean       16.673194
std         5.901662
min         1.000000
25%        13.000000
50%        18.000000
75%        21.000000
max        25.000000
Name: income06, dtype: float64
```

```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn import linear_model
```

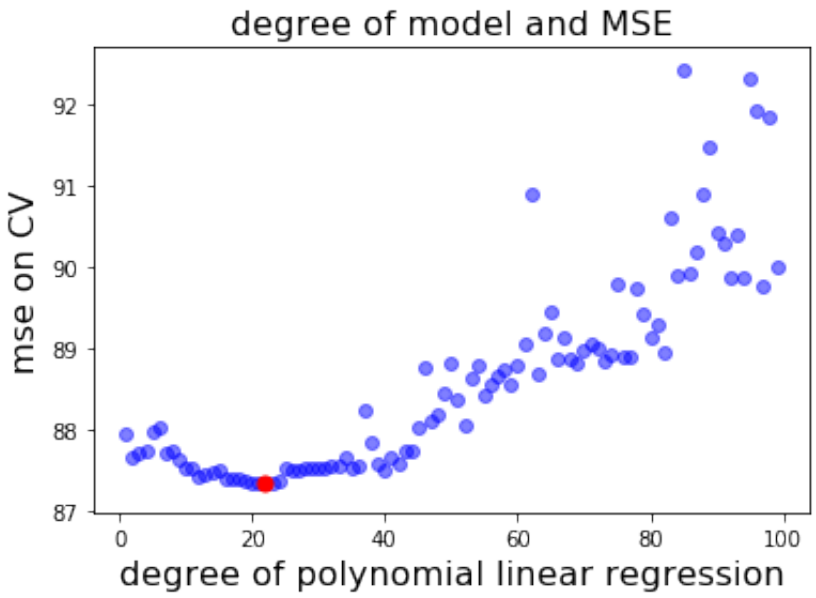
Q1.polynomial regression

```
indice = np.random.permutation(np.array(list(range(len(df_train)))))
data_array = np.array(df_train[['egalit_scale','income06']])

kf = KFold(n_splits=10)
degree_mse = {}
for d in range(1,100):
    MSE = []
    for train_indice, test_indice in kf.split(indice):
        x_train = data_array[train_indice,1]
        y_train = data_array[train_indice,0]
        x_test = data_array[test_indice,1]
        y_test = data_array[test_indice,0]
        poly = PolynomialFeatures(degree = d)
        linear = linear_model.LinearRegression(fit_intercept = True)
        xtrain_poly = poly.fit_transform(x_train.reshape(-1,1))
        xtest_poly = poly.fit_transform(x_test.reshape(-1,1))

        #poly.fit(xtrain_poly,y_train)
        linear.fit(xtrain_poly,y_train)
        MSE.append(mean_squared_error(y_test, linear.predict(xtest_poly)))
    degree_mse[d] = np.mean(MSE)
```

```
sorted(degree_mse.items(),key=lambda x:x[1])[0]
```

```
(22, 87.33369635078013)
```

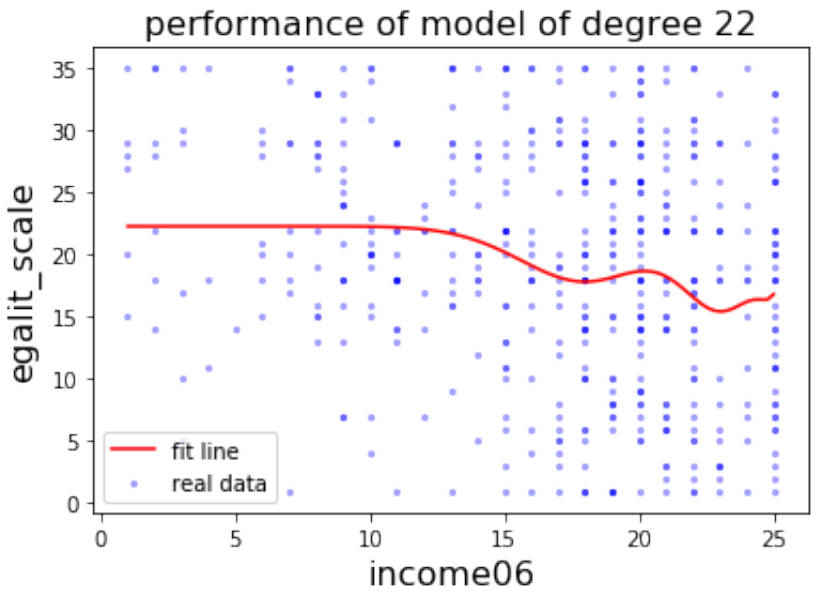
```
plt.scatter(list(degree_mse.keys()),list(degree_mse.values()),c='b',alpha =
0.5)
plt.scatter(22,degree_mse[22],c = 'r', s = 50, alpha = 1)
plt.xlabel('degree of polynomial linear regression',size = 16)
plt.ylabel('mse on CV',size = 16)
plt.title('degree of model and MSE',size = 16)
plt.show()
```




```
# mse on test set:
data_test = np.array(df_test[['egalit_scale','income06']])
x_test = data_test[:,1]
y_test = data_test[:,0]
poly = PolynomialFeatures(degree = 22)
linear = linear_model.LinearRegression(fit_intercept = True)
xtest_poly = poly.fit_transform(x_test.reshape(-1,1))

linear.fit(xtest_poly,y_test)
mse_test = (mean_squared_error(y_test, linear.predict(xtest_poly)))
print(f'mse of model of degree 22 ont test set: {mse_test}')
```

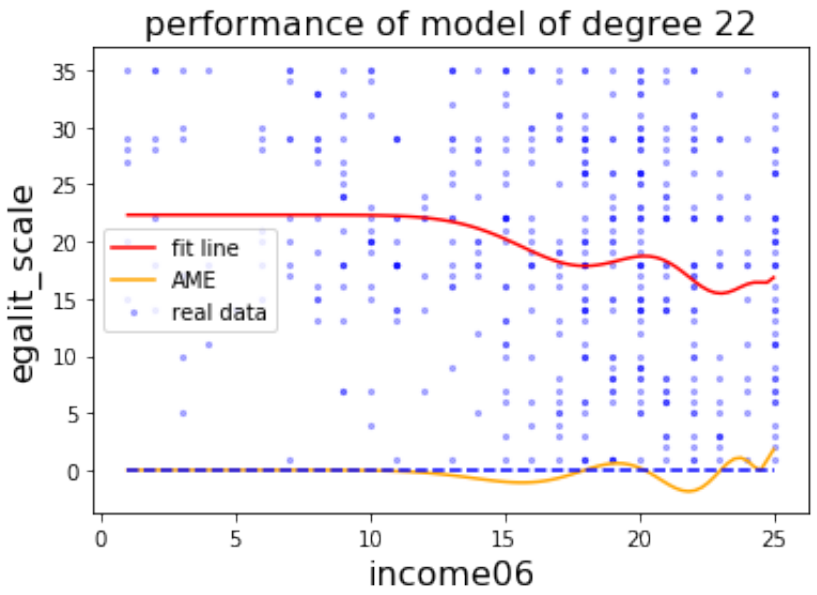
```
mse of model of degree 22 ont test set: 85.69060318782633
```

```
x_range = np.linspace(np.min(x_test),np.max(x_test),100)
x_range_poly = poly.fit_transform(x_range.reshape(-1,1))
y_fit = linear.predict(x_range_poly)
plt.plot(x_range,y_fit,c= 'r',label = 'fit line')
plt.xlabel('income06',size = 16)
plt.ylabel('egalit_scale',size = 16)
plt.scatter(x_test,y_test,c = 'b',s = 5, alpha = 0.3,label = 'real data')
plt.title('performance of model of degree 22',size = 16)
plt.legend()
plt.show()
```




```
## AME
```

```
x_range = np.linspace(np.min(x_test),np.max(x_test),100)
delta_x = x_range[1] - x_range[0]
x_range_poly = poly.fit_transform(x_range.reshape(-1,1))
y_fit = linear.predict(x_range_poly)
ame = np.gradient(y_fit,delta_x)
plt.plot(x_range,y_fit,c= 'r',label = 'fit line')
plt.plot(x_range,ame,c='orange',label = 'AME')
plt.plot(x_range, [0]*len(x_range),'--',c = 'blue')
plt.xlabel('income06',size = 16)
plt.ylabel('egalit_scale',size = 16)
plt.scatter(x_test,y_test,c = 'b',s = 5, alpha = 0.3,label = 'real data')
plt.title('performance of model of degree 22',size = 16)
plt.legend()
plt.show()
```



interpretation of the AME

As the income increases, the feeling of egalitarianism does not change at first, but when the income is larger than 13, the egalitarianism will change as well. In general, as income increases, the feeling of egalitarianim will decrease, but the decreasing is not monotonically.

Q2.step function

```
def construct_step_function_variable(x, n_cuts):
    x_range = np.array(x)
    #k_length = (np.max(x_range) - np.min(x_range))/ (n_cuts + 1)
    k_length = (25 - 1)/ (n_cuts + 1)
    response = np.zeros((len(x_range),n_cuts))
    for i in range(len(x_range)):
        if x_range[i]//k_length == 0:
            continue
        response[i][int(min(x_range[i]//k_length-1,n_cuts-1))] = 1
    return np.linspace(np.min(x_range), np.max(x_range), n_cuts+2), response
```

```
indice = np.random.permutation(np.array(list(range(len(df_train)))))
data_array = np.array(df_train[['egalit_scale','income06']])

kf = KFold(n_splits=10)
degree_mse = {}
for n_cuts in range(1,34):
    MSE = []
    for train_indice, test_indice in kf.split(indice):
        x_train = data_array[train_indice,1]
        y_train = data_array[train_indice,0]
        x_test = data_array[test_indice,1]
        y_test = data_array[test_indice,0]
        _,x_train_sf = construct_step_function_variable(x_train, n_cuts)
```

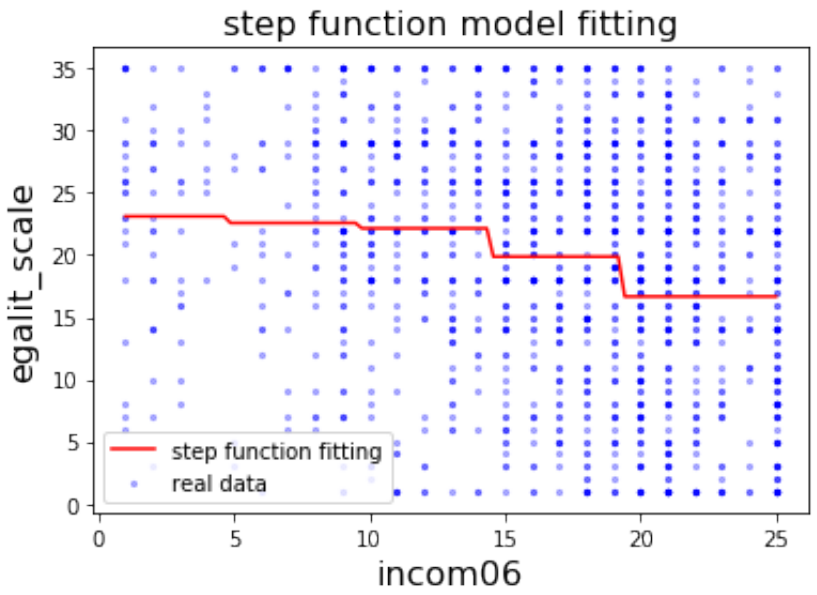
```
        _,x_test_sf = construct_step_function_variable(x_test, n_cuts)
        linear = linear_model.LinearRegression(fit_intercept = True)

        #poly.fit(xtrain_poly,y_train)
        linear.fit(x_train_sf,y_train)
        MSE.append(mean_squared_error(y_test,linear.predict(x_test_sf)))
    degree_mse[n_cuts] = np.mean(MSE)
```

the best number of cuts:

```
sorted(degree_mse.items(),key=lambda x:x[1])[0]
```

```
(4, 87.701579071394903)
```

```
## the best cut is 4:
x_train = data_array[:,1]
y_train = data_array[:,0]
ck,x_train_sf = construct_step_function_variable(x_train, 4)
linear = linear_model.LinearRegression(fit_intercept = True)
linear.fit(x_train_sf,y_train)
x_range = np.linspace(1,25,100)
_,x_range_train = construct_step_function_variable(x_range, 4)
y_predict = linear.predict(x_range_train)
plt.scatter(x_train,y_train,c='b', label = 'real data',alpha = 0.3, s=5)
plt.plot(x_range,y_predict,c='r', label = 'step function fitting')
plt.xlabel('incom06',size = 16)
plt.ylabel('egalit_scale',size = 16)
plt.title('step function model fitting', size = 16)
plt.legend()

plt.show()
```




```
linear.coef_
```

```
array([-0.52985075, -0.96216992, -3.24104149, -6.4472338 ])
```

interpret the result:

Step functions give us the results that when income of a person increase, the feeling towards egalitarianism is decreasing. As income increases from 0 to 5/10/15/20, the feeling towards egalitarianism will decrease 0.53/0.96/3.24/6.45.

Q3.natural regression spline

```
from patsy import dmatrix
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

```
/Users/apple/anaconda3/lib/python3.6/site-
packages/statsmodels/compat/pandas.py:56: FutureWarning: The
pandas.core.datetools module is deprecated and will be removed in a future
version. Please use the pandas.tseries module instead.
  from pandas.core import datetools
```

```
indice = np.random.permutation(np.array(list(range(len(df_train)))))
```

```
data_array = np.array(df_train[['egalit_scale','income06']])

kf = KFold(n_splits=10)
degree_mse = {}
for d in range(0,10):
    MSE = []
    for train_indice, test_indice in kf.split(indice):
        x_train = data_array[train_indice,1]
        y_train = data_array[train_indice,0]
        x_test = data_array[test_indice,1]
        y_test = data_array[test_indice,0]
        transformed_x_train = \
        dmatrix(f"bs(train, knots=(5,10,15,20), degree={d},
include_intercept=False)", {"train": x_train},return_type='dataframe')
        transformed_x_test = \
        dmatrix(f"bs(validation, knots=(5,10,15,20), degree={d},
include_intercept=False)", {"validation": x_test},return_type='dataframe')

        linear = linear_model.LinearRegression(fit_intercept = True)

        #poly.fit(xtrain_poly,y_train)
        linear.fit(transformed_x_train,y_train)

MSE.append(mean_squared_error(y_test,linear.predict(transformed_x_test)))
    degree_mse[d] = np.mean(MSE)
```

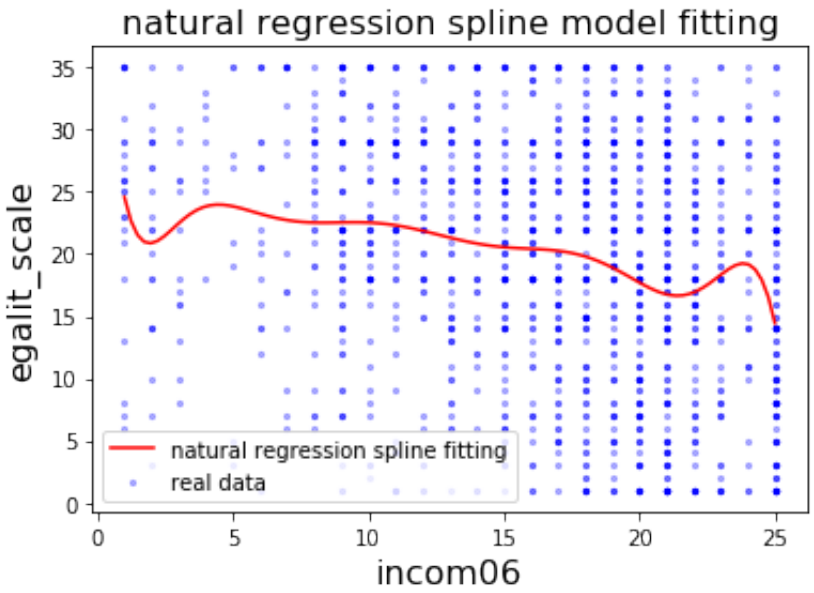
the degree chose is 5, with cv mse 87.67

```
sorted(degree_mse.items(),key=lambda x:x[1])[0]
```

```
(5, 87.674707303368081)
```

```
## the best degree is 5:
d = 5
x_train = data_array[:,1]
y_train = data_array[:,0]
transformed_x_train = \
dmatrix(f"bs(train, knots=(5,10,15,20), degree={d}, include_intercept=False)",
{"train": x_train},return_type='dataframe')
linear = linear_model.LinearRegression(fit_intercept = True)
linear.fit(transformed_x_train,y_train)
x_range = np.linspace(1,25,100)
transformed_x_range = \
```

```
dmatrix(f"bs(train, knots=(5,10,15,20), degree={d}, include_intercept=False)",
{"train": x_range},return_type='dataframe')
y_predict = linear.predict(transformed_x_range)
plt.scatter(x_train,y_train,c='b', label = 'real data',alpha = 0.3, s =5)
plt.plot(x_range,y_predict,c='r', label = 'natural regression spline fitting')
plt.xlabel('incom06',size = 16)
plt.ylabel('egalit_scale',size = 16)
plt.title('natural regression spline model fitting', size = 16)
plt.legend()

plt.show()
```



Q4.egalitarianism and everything

```
available_variables = list(df_train.describe().columns)
```

```
available_variables
```

```
['age',
 'authoritarianism',
 'childs',
 'con_govt',
 'egalit_scale',
 'income06',
 'science_quiz',
 'sibs',
 'social_connect',
 'tolerance',
 'tvhours',
 'wordsum']
```

```
# data processing
from sklearn import preprocessing
data_array = copy.copy(df_train[available_variables])
data_array.drop(['egalit_scale'],axis = 1)
X_train = np.array(data_array)
Y_train = np.array(df_train['egalit_scale'])
X_scaled = preprocessing.scale(X_train)
# hyperparameter tuning
# fitting
```

```
/Users/apple/anaconda3/lib/python3.6/site-
packages/sklearn/utils/validation.py:475: DataConversionWarning: Data with
input dtype int64 was converted to float64 by the scale function.
  warnings.warn(msg, DataConversionWarning)
```

linear regression

```
## linear model:
indice = np.random.permutation(np.array(list(range(len(df_train)))))
MSE = []
kf = KFold(n_splits=10)
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled[test_indice]
    y_test = Y_train[test_indice]
    linear = linear_model.LinearRegression(fit_intercept = True)
    linear.fit(x_train,y_train)
    MSE.append(mean_squared_error(y_test, linear.predict(x_test)))
```

the mse of linear model on cv

```
## estimate the MSE of linear model
np.mean(MSE)
```

```
3.0755783583731145e-28
```

elastic net regression

```
## elastic net model
from sklearn.linear_model import ElasticNet
indice = np.random.permutation(np.array(list(range(len(df_train)))))
kf = KFold(n_splits=10)
MSE_list = dict()
for ratio in list(np.linspace(0,1,11)):# the ratio is between 0 and 1,
suggesting the degree of L1 penalty, 1= 100% L1
    for lambdaa in list(np.linspace(0.01,0.55,12)):# the penalty degree on the
complexity of model
        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,max_iter=10000)
            clf.fit(X_scaled[train_indice],Y_train[train_indice])

mse_list.append(mean_squared_error(Y_train[test_indice],clf.predict(X_scaled[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)
```

```
sorted(MSE_list.items(),key=lambda x:x[1])[0]
```

```
((1.0, 0.01), 0.00010063917589651194)
```

```
## the alpha is 0.01 and the l1_ratio is 1.0
```

```
indice = np.random.permutation(np.array(list(range(len(df_train)))))
MSE = []
kf = KFold(n_splits=10)
for train_indice, test_indice in kf.split(indice):
    x_train = X_scaled[train_indice]
    y_train = Y_train[train_indice]
    x_test = X_scaled[test_indice]
    y_test = Y_train[test_indice]
    clf = ElasticNet(alpha=0.01,l1_ratio= 1.0,max_iter=10000)
    clf.fit(x_train, y_train)
    MSE.append(mean_squared_error(y_test, clf.predict(x_test)))
```

the mse of elastic net regression on cv

```
## the MSE of elastic net:
np.mean(MSE)
```

```
0.00010063917589651194
```

PCR

```
## PCR:
from sklearn.decomposition import PCA
pca = PCA()
x_reduced= pca.fit_transform(X_scaled)
```

```
x_reduced.shape
```

```
(1481, 12)
```

```
# the variance the PCA interpreted:
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

```
array([ 22.91,  36.07,  45.48,  54.08,  61.64,  68.62,  74.98,  80.59,
        86.04,  91.35,  95.93,  99.99])
```

```
indice = np.random.permutation(np.array(list(range(len(df_train)))))
kf = KFold(n_splits=10)
degree_mse = {}
for n_comps in range(1,13):
    MSE = []
    for train_indice, test_indice in kf.split(indice):
        x_train = X_scaled[train_indice]
        y_train = Y_train[train_indice]
        x_test = X_scaled[test_indice]
        y_test = Y_train[test_indice]
        linear_pcr = linear_model.LinearRegression(fit_intercept = True)
        linear_pcr.fit(x_train[:,:n_comps],y_train)

MSE.append(mean_squared_error(y_test,linear_pcr.predict(x_test[:,:n_comps])))
    degree_mse[n_comps] = np.mean(MSE)
```

the mse of PCR on cv

```
sorted(degree_mse.items(),key=lambda x:x[1])[0]
```

```
(8, 9.8944463770616831e-29)
```

PLS

```
## PLS
from sklearn.cross_decomposition import PLSRegression

indice = np.random.permutation(np.array(list(range(len(df_train)))))

kf = KFold(n_splits=10)
degree_mse = {}
for n_comps in range(1,13):
    MSE = []
    for train_indice, test_indice in kf.split(indice):
        x_train = X_scaled[train_indice]
        y_train = Y_train[train_indice]
        x_test = X_scaled[test_indice]
        y_test = Y_train[test_indice]
```

```
        pls = PLSRegression(n_components=n_comps, scale=False)
        pls.fit(x_train,y_train)
        MSE.append(mean_squared_error(y_test,pls.predict(x_test)))
    degree_mse[n_comps] = np.mean(MSE)
```

```
/Users/apple/anaconda3/lib/python3.6/site-
packages/sklearn/cross_decomposition/pls_.py:287: UserWarning: Y residual
constant at iteration 11
  warnings.warn('Y residual constant at iteration %s' % k)
```

the mse of pls on cv

```
sorted(degree_mse.items(),key=lambda x:x[1])[0]
```

```
(12, 1.1473716274184564e-20)
```

Q5. feature importance

```
## the belowing is the code of R:
"""
library(glmnet)
library(caret)
library(iml)
library(plotly)
library(margins)
df_train = read_csv("/Users/apple/Desktop/gss_train.csv")
y_train <- df_train['egalit_scale']
x_train <-
df_train[c('egalit_scale','age','authoritarianism','childs','con_govt','income0
6','science_quiz','sibs','social_connect','tolerance','tvhours','wordsum')]
x_scaled <- scale(x_train)
x_scaled_df <- as.data.frame((x_scaled), row.names=
c('egalit_scale','age','authoritarianism','childs','con_govt','income06','scien
ce_quiz','sibs','social_connect','tolerance','tvhours','wordsum'))
features <-
x_scaled_df[c('age','authoritarianism','childs','con_govt','income06','science_
quiz','sibs','social_connect','tolerance','tvhours','wordsum')]
cv_10 = trainControl(method = "cv", number = 10)
Glm <- lm(egalit_scale ~., data = x_scaled_df)
Glmnet = train(
  egalit_scale ~., data = x_scaled_df,
```

```
  method = "glmnet",
  trControl = cv_10
)
Pcr <- train(
  egalit_scale ~., data = x_scaled_df,
  method = "pcr",
  trControl = cv_10
)
Pls <- train(
  egalit_scale ~., data = x_scaled_df,
  method = "pls",
  trControl = cv_10
)

# construct the predictor
predictor.glm <- Predictor$new(
  model = Glm,
  data = features,
  y = y_train
)
predictor.glmnet <- Predictor$new(
  model = Glmnet,
  data = features,
  y = y_train
)
predictor.pcr <- Predictor$new(
  model = Pcr,
  data = features,
  y = y_train
)
predictor.pls <- Predictor$new(
  model = Pls,
  data = features,
  y = y_train
)

interact.glm <- Interaction$new(predictor.glm) %>%
  plot() +
  ggtitle("glm") +
  theme_minimal(base_size = 12)

interact.glmnet  <- Interaction$new(predictor.glmnet) %>%
  plot() +
  ggtitle("glmnet") +
  theme_minimal(base_size = 12)

interact.pcr <- Interaction$new(predictor.pcr) %>%
  plot() +
  ggtitle("pcr") +
```

```
  theme_minimal(base_size = 12)

interact.pls <- Interaction$new(predictor.pls) %>%
  plot() +
  ggtitle("pls") +
  theme_minimal(base_size = 12)
# plot

grid.arrange(interact.glm, interact.glmnet, interact.pcr, interact.pls, ncol =
2)
"""
```

At first we can see that most variables have relatively low interaction effects with other variables, so the variables are kind of independent of each other. Almost all models choose age as the feature of most strongest overall interaction strength with other features, and income and authoritarianism are picked as the features of strongest overall interaction strength by GLMNET and PLS, respectively, but at the same time, PCR gave low interaction strength to authoritarianism. In conclusion, the interaction effects is not very consistent across different models.