In [17]:
```python
import sklearn
from sklearn.base import BaseEstimator
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.cross_decomposition import PLSRegression, PLSSVD
from sklearn.linear_model import LinearRegression as lr
from sklearn.linear_model import ElasticNetCV, Ridge
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PolynomialFeatures as pf
from sklearn.preprocessing import LabelEncoder as le
from sklearn.preprocessing import KBinsDiscretizer as kb
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import make_scorer
from sklearn.pipeline import make_pipeline, Pipeline


from collections import Counter
import tabulate as tb

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.colors
import seaborn

import scipy as sp
from scipy.interpolate import CubicSpline, UnivariateSpline
```

```python
import collections
import os
import os.path
import random
import re
import glob
import pandas as pd
import requests
import json
import math


from patsy import dmatrix



%matplotlib inline
```

In [9]:
```python
gss_tr = pd.read_csv('gss_train.csv')
gss_te = pd.read_csv('gss_test.csv')
```

In [59]:
```python
np.random.seed(51)
```

In [60]:
```python
#gss_tr = pd.read_csv('gss_train.csv')
#gss_te = pd.read_csv('gss_test.csv')

assert set(gss_te.columns) == set(gss_tr.columns)

collect = {}

for col in gss_te.columns:
    pred_test, pred_train = list(gss_te[col]), list(gss_tr[col])
    label_e = le().fit(pred_test+pred_train)
    gss_te[col] = label_e.transform(pred_test)
    gss_tr[col] = label_e.transform(pred_train)
    collect[col] = label_e.classes_
```

In [ ]:
```
gss_te, gss_tr = gss_te.dropna(axis=0), gss_tr.dropna(axis=0)
```

In [ ]:
```
x_train, y_train = gss_tr.drop('egalit_scale', axis=1), gss_tr['egalit_
scale']
x_test, y_test = gss_te.drop('egalit_scale', axis=1), gss_te['egalit_sc
ale']
```

In [ ]:
```
x_train, y_train, x_test, y_test = [i.to_numpy() for i in [x_train, y_t
rain, x_test, y_test]]
y_train, y_test = (i.reshape(-1, 1) for i in (y_train, y_test))
```

In [61]:
```
income_train, income_test = (np.array(i).reshape(-1, 1) for i in (gss_t
r['income06'], gss_te['income06']))
```

Egalitarianism and income

(20 points) Perform polynomial regression to predict egalit_scale as a function of income06. Use and plot 10-fold cross-validation to select the optimal degree d for the polynomial based on the MSE. Plot the resulting polynomial fit to the data, and also graph the average marginal effect (AME) of income06 across its potential values. Be sure to provide substantive interpretation of the results.

In [ ]:
```
poly_pipe =[('poly', pf()), ('lr', lr())]
p_reg = Pipeline(poly_pipe)

params = {'poly__degree': range(1, 51)}

scores = make_scorer(MSE, greater_is_better=False) #sklearn
```

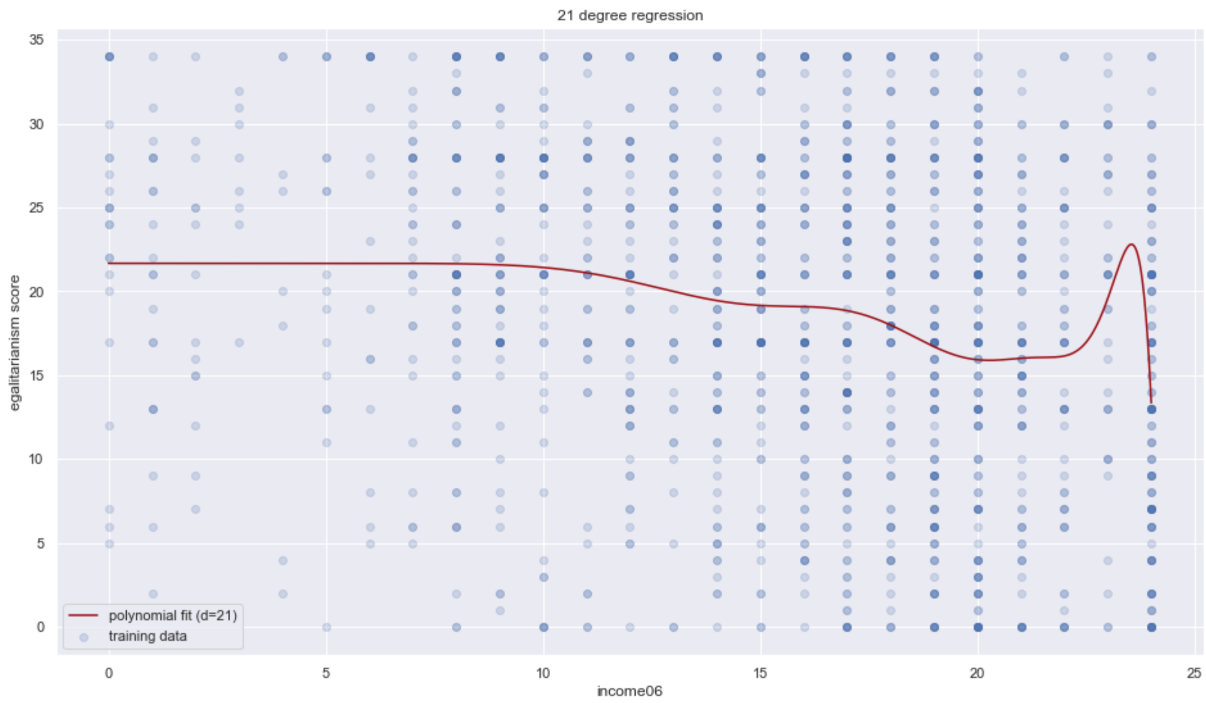
In [ ]:
```
cv = GridSearchCV(p_reg, params, scoring=scores, cv=10)
cv.fit(income_train, y_train)
```

In [63]:
```
res = pd.DataFrame(cv.cv_results_)
best = cv.best_estimator_
```

```
best_d = cv.best_params_['poly__degree']
```

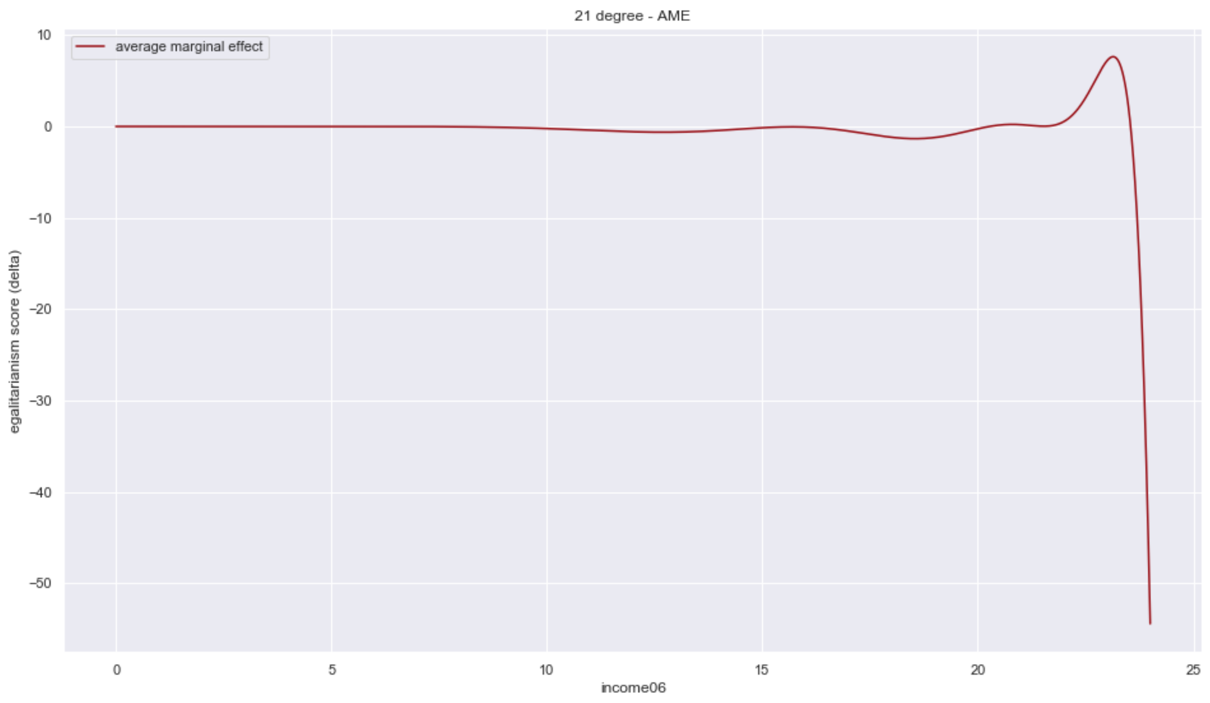
In [106]:

```
income_plot = np.linspace(income_train.min(), income_train.max(), 10000
).reshape(-1, 1)
plot_y = best.predict(income_plot)

plt.scatter(income_train, y_train, label='training data', alpha=0.2)
plt.plot(income_plot, plot_y, label=f'polynomial fit (d={best_d})', c=p
lt.cm.Maroons(0.9))

plt.legend()

plt.title(f'{best_d} degree regression')
plt.xlabel('income06')
plt.ylabel('egalitarianism score')
```



In [108]:
```
sp = float(income_plot[1]-income_plot[0])
plot_ame = np.gradient(plot_y.reshape(-1), sp)
plt.plot(income_plot, plot_ame, label=f'average marginal effect', c=plt
.cm.Maroons(0.9))
plt.legend()

plt.title(f'{best_d} degree - AME')
plt.xlabel('income06')
plt.ylabel('egalitarianism score (delta)')
```

Out[108]:
```
Text(0, 0.5, 'egalitarianism score (delta)')
```



(20 points) Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.

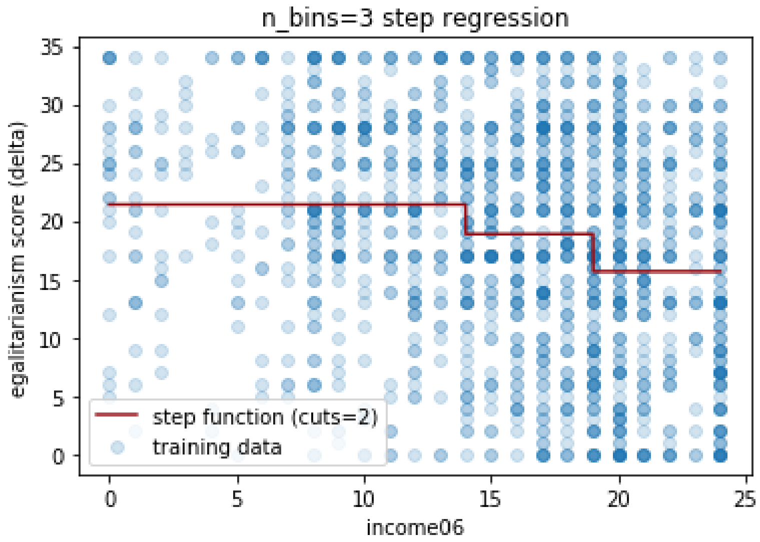
In [43]:
```
step_pipe = Pipeline([('cut', kb()), ('lr', lr())])
params = {'cut__n_bins': range(2, 20)} #can't do 1, threw error

cv = GridSearchCV(step_pipe, params, n_jobs=-1, scoring='neg_mean_squar
ed_error', cv=10)
```

In [44]:
```
cv.fit(income_train, y_train)
best = cv.best_estimator_
best_bins = cv.best_params_['cut__n_bins']
```

In [53]:
```
income_plot = np.linspace(income_train.min(), income_train.max(), 10000
).reshape(-1, 1)
```

```
plot_y = best.predict(income_plot)

plt.scatter(income_train, y_train, label='training data', alpha=0.2)
plt.plot(income_plot, plot_y, label=f'step function (cuts={best_cuts-1})', c=plt.cm.Reds(0.9))
plt.title(f'n_bins={best_bins} step regression')
plt.legend()
plt.xlabel('income06')
plt.ylabel('egalitarianism score (delta)')
```

Out[53]: Text(0, 0.5, 'egalitarianism score (delta)')



(20 points) Fit a natural regression spline to predict egalit_scale as a function of income06. Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the results of the optimal model.

In [54]:
```
y_train = gss_tr['egalit_scale']
x_train = gss_tr['income06'].values.reshape(-1, 1)
y_test = gss_te['egalit_scale']
x_test = gss_te['income06'].values.reshape(-1, 1)
```

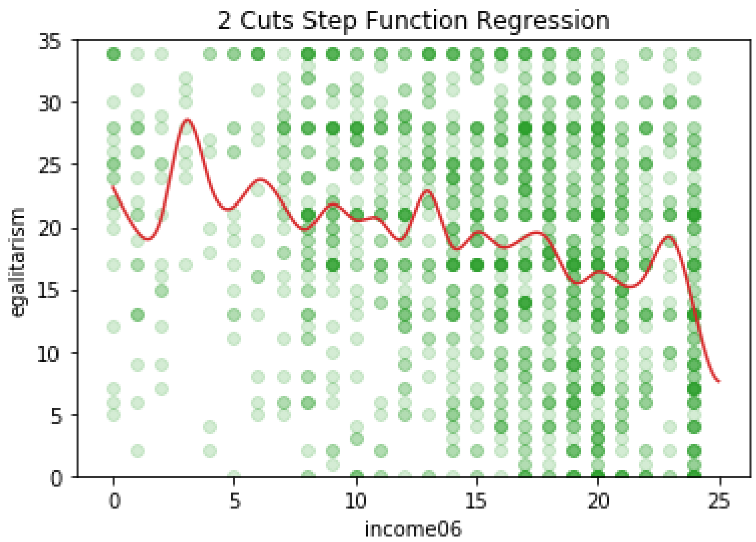
In [58]:

```
track = {'mean_squared_error': [], 'mods': [], 'k': []}

cv_10 = KFold(n_splits=10)
splits = cv_10.split(x_train, y_train)

for train, test in splits:
    x_tr_n = x_train[train].flatten()
    y_tr_n = y_train[train].values.flatten()
    x_flat = x_train[test].flatten()
    y_flat = y_train[test].values.flatten()
    res = pd.DataFrame(data={'feat':x_tr_n, 'predict':y_tr_n}, index=pd
.RangeIndex(len(x_tr_n)))

    res = res.groupby('feat', as_index=False)['predict'].mean()
    x_ch = res['feat'].values
    y_ch = res['predict'].values

    spline = CubicSpline(x_ch, y_ch, bc_type='natural')

    y_egal = sp(x_flat)

    mean_squared_error = np.mean((y_egal - y_flat) ** 2)
    print(mean_squared_error)
    track['mean_squared_error'].append(mean_squared_error)
    track['mods'].append(spline)

best = np.argmin(track['mean_squared_error'])
print(best)
best = track['mods'][best]

plot_x = np.linspace(0, 25, 10000).reshape(-1, 1)
plot_y = best(plot_x)
```

```
88.90076400132222
83.95884373090617
88.77919769150931
86.40146010254689
85.50891974424688
```

```
101.57410227823715
86.44463506276448
97.27130464359588
89.82594483834163
75.9863648235786
9
```

In [57]:
```
plt.plot(plot_x, plot_y, c='C3')
plt.scatter(x_train, y_train, c='C2', alpha=.2)
plt.ylim(0, 35)
plt.xlabel('income06')
plt.ylabel('egalitarism')
plt.title('2 Cuts Step Function Regression')
plt.show()
```



In [ ]:

Egalitarianism and everything

(20 points total) Estimate the following models using all the available predictors (be sure to perform appropriate data pre-processing (e.g., feature standardization) and hyperparameter tuning (e.g. lambda for PCR/PLS, lambda and alpha for elastic net). Also use 10-fold cross-validation for each model to estimate the model's performance using MSE): a. (5 points) Linear regression b. (5 points) Elastic net regression c. (5 points) Principal component regression d. (5 points) Partial least squares regression

a)

In [67]:
```
x1, x2 = StandardScaler(), StandardScaler()
x1, x2 = x1.fit(x_tr), x2.fit(x_te)
xtr_s, xte_s = x1.transform(x_tr), x2.transform(x_te)
print(x_tr.shape, y_tr.shape)
#print(x_te.shape, y_te.shape)
```

```
(1481, 44) (1481, 1)
```

In [68]:
```
lrcv = GridSearchCV(lr(), {}, scoring='neg_mean_squared_error', cv=10)
lrcv.fit(xtr_s, y_tr)
best_lr = lrcv.best_estimator_
lr_err = MSE(y_te, best_lr.predict(xte_s))
```

In [69]:
```
print(f"Best linear regression error: {lr_err}")
```

```
Best linear regression error: 63.928057088260516
```

b)

In [70]:
```
elcv = ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1], n_alphas=10
, cv=10)
y_tr = y_tr.reshape(-1,)
elcv.fit(xtr_s, y_tr)
el_err = MSE(y_te, elcv.predict(xte_s))
print(f"Best ElasticNet error: {el_err}\n\nParameters:\nlambda = {elcv.
alpha_}\nalpha = {elcv.l1_ratio_}")
```

```
Best ElasticNet error: 62.56902435370069

Parameters:
lambda = 0.19753166246833653
alpha = 0.5
```

c)

In [71]:
```
pcr = Pipeline([('pca', PCA()), ('ridge', Ridge())])
param_grid = {'pca__n_components':np.arange(2, 24, 2), 'ridge__alpha':[
0.01, 0.05]+list(np.arange(0.1, 1, 10))}
pcacv = GridSearchCV(pcr, param_grid, scoring='neg_mean_squared_error',
 cv=10, refit=True)
pcacv.fit(xtr_s, y_tr)
best_pca = pcacv.best_estimator_
best_n = pcacv.best_params_['pca__n_components']
best_lambda = pcacv.best_params_['ridge__alpha']
pca_err = MSE(y_te, best_pca.predict(xte_s))
print(f"Best PCR error: {pca_err}\nParameters:\nn_components = {best_
n}\nlambda = {best_lambda}")
```

```
Best PCR error: 62.32194871839432

Parameters:
n_components = 22
lambda = 0.05
```

d)

In [72]:
```
pls = PLSRegression()
plscv = GridSearchCV(pls, param_grid={'n_components':np.arange(2, 21, 2
)}, scoring='neg_mean_squared_error', cv=10)
plscv.fit(xtr_s, y_tr)
best_pls = plscv.best_estimator_
```

In [74]:
```
best_n = plscv.best_params_['n_components']
#best_lambda = plscv.best_params_['alpha']
```

```
pls_err = MSE(y_te, best_pls.predict(xte_s))
print(f"Best PCR error: {pls_err}\n\nParameters:\nn_components = {best_
n}")#\nlambda = {best_lambda}")
```

```
Best PCR error: 63.927709350623815

Parameters:
n_components = 12
```

1. (20 points) For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other (e.g., talk about feature importance, conditional effects, how these are ranked differently across different models, etc.).

In [99]:
```
from mlxtend.evaluate import feature_importance_permutation
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean',verb
ose=0)
imputer = imputer.fit(x_test)
x_test = imputer.transform(x_test)
seaborn.set(rc={'figure.figsize':(16,9)})
```

In [101]:
```
def plot_imp(model):
    imp_vals, _ = feature_importance_permutation(
    predict_method=model.predict,
    X=x_test,
    y=y_test,
    metric='r2', # r2 metrics is recommended for regressors
    num_rounds=10)

    col = []
    imp = []

    for i in range(x_test.shape[1]):
        col.append(gss_te.columns[i])
        imp.append(imp_vals[i])
```

```
    seaborn.set_style("white")
    ax = sns.barplot(x=imp, y=col, color='lightblue').set_title("R2 of
Features")

    return ax
```

In [102]:
```
plt.figure(figsize=(12, 12))
lm_plot = plot_imp(lm)
plt.title('Linear Regression');
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-102-d6a8a08fc0fd> in <module>
      1 plt.figure(figsize=(12, 12))
----> 2 lm_plot = plot_imp(lm)
      3 plt.title('Linear Regression');

NameError: name 'lm' is not defined

<Figure size 864x864 with 0 Axes>
```

In [ ]: