In [17]:

```
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
```

```
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]:
```
gss_tr = pd.read_csv('gss_train.csv')
gss_te = pd.read_csv('gss_test.csv')
```

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

In [60]:
```
#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']
```

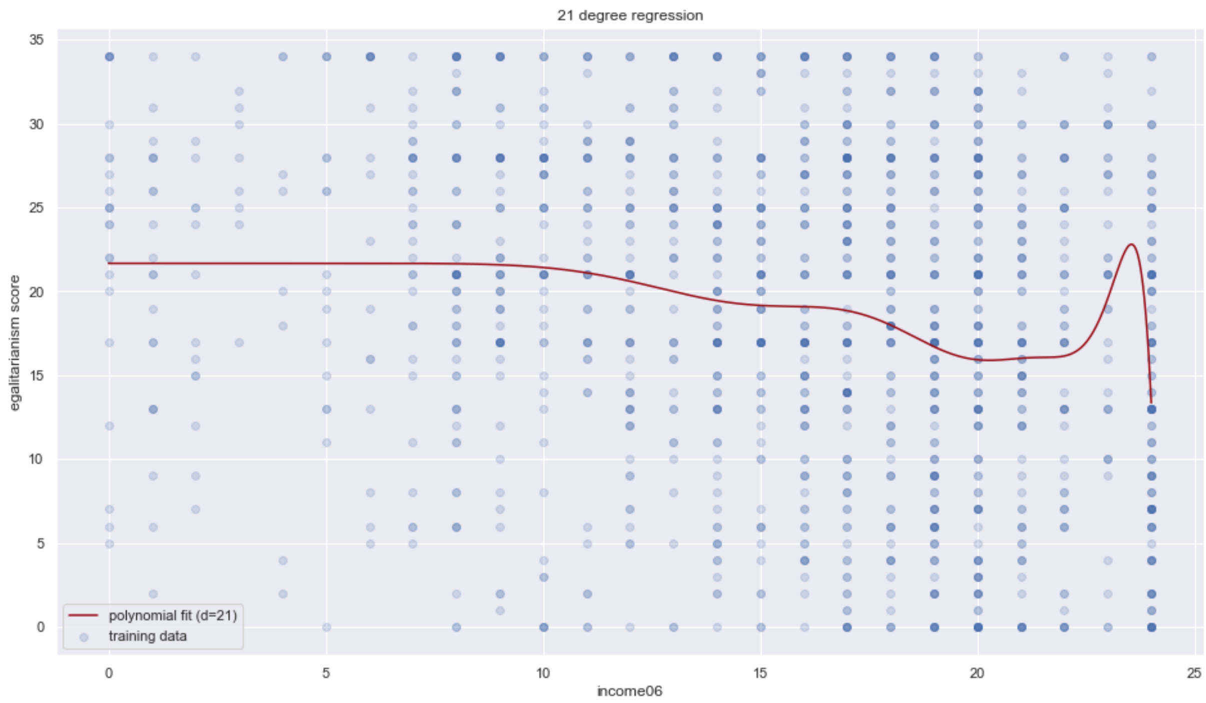
In [106]:

```
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')
```



- I ran the polynomial model for degrees ranging from 1-50, and the lowest MSE value corresponded to a degree of 21, for 10-fold cross validation . However, we can see from the plot that this curve does not fit the data very well.

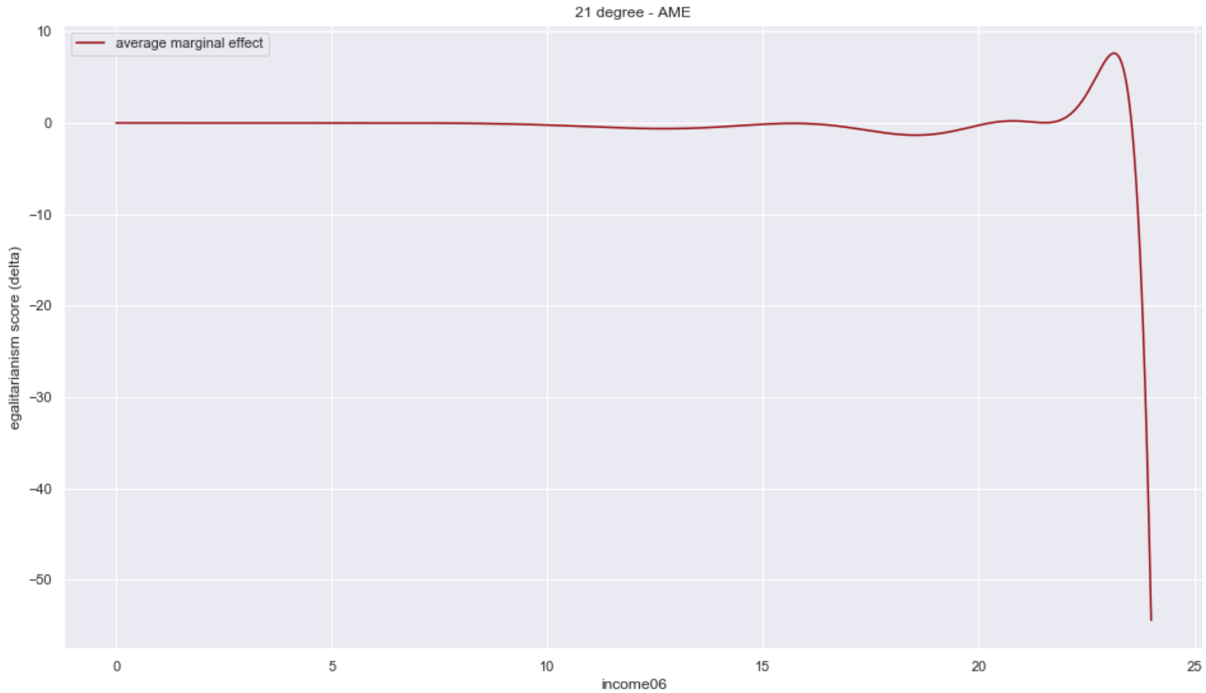
In [108]:

```
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)')



- The MSE for degree = 21 was roughly 80, and the AME is zero up until roughly degree = 23 (suggests little to no effect). Around this point, we can see that egalitarism score decreases as income goes up. However, as mentioned previously, the polynomial form (despite varying degrees) does not fit this data very well.
- The computed AME = roughly -.36, though that could be explained by the suggest shift in trajectory described.

(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']
```

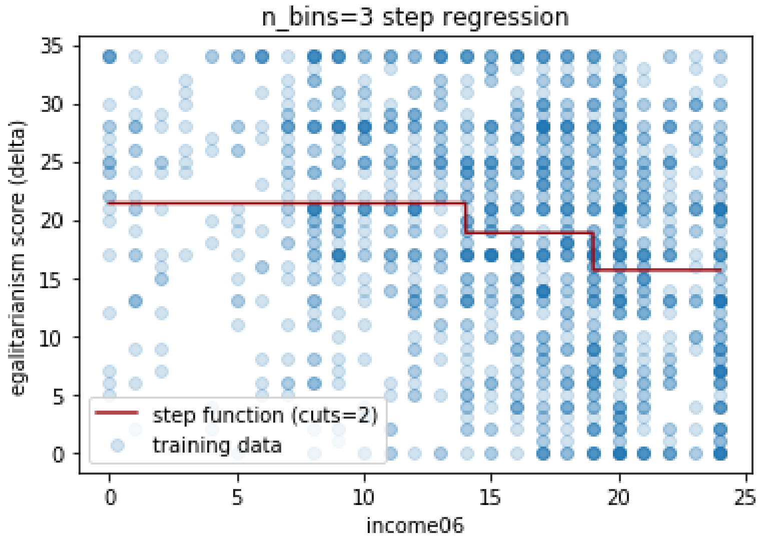
In [53]:
```
income_plot = np.linspace(income_train.min(), income_train.max(), 10000
).reshape(-1, 1)

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)')

#MSE:
#89.90076400132222
#84.95884373090617
#88.77919769150931
#87.40146010254689
#84.50891974424688
#102.57410227823715
#85.44463506276448
#98.27130464359588
#90.82594483834163
#89.9863648235786
```

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



- We can see that 3 bins = the optimal number of cuts (2). The MSE for 3 bins is still quite high (~89), and quite high across the board (see previous cell) across the range of cuts that I chose (across 10-fold cv), suggesting that a step function might not fit our data well (it's an overly restrictive approach for data of this spread). We can also see that there seem to be trends that we can isolate, bin to bin. In other words, egalitarian scores seem to vary across the three bins - it differs across low, medium, and high income.

(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)

#MSE printed below:
```

```
88.90076400132222
83.95884373090617
88.77919769150931
86.40146010254689

85.50891974424688
```

```
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()
```

- The MSE is quite high here as well - coming in at around 89. The optimal # of cuts is 2 here, like we obtained using the step function. The flexible nature of the spline tells us more about how income and egalitarianism vary across all three bins, but we can see that despite smoothing, this model does not fit the data very well.

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]:
```
x_1, x_2 = StandardScaler(), StandardScaler()
x_1, x_2 = x_1.fit(x_tr), x_2.fit(x_test)
x_train_shape, x_test_shape = x_1.transform(x_train), x_2.transform(x_t
est)
```

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

In [68]:
```
linreg = GridSearchCV(lr(), {}, scoring='neg_mean_squared_error', cv=10
)
linreg.fit(x_train_shape, y_tr)
best_linreg = linreg.best_estimator_
linreg_err = MSE(y_test, best_linreg.predict(x_test_shape))

#Best linear regression MSE: 63.928057088260516
```

b)

In [ ]:
```
elnet = ElasticNetCV(l1_ratio=[.1, .5, .7, .9, .95, .99, 1], n_alphas=1
0, cv=10)
y_train = y_train.reshape(-1,)
elnet.fit(x_train_shape, y_train)
elnet_err = MSE(y_test, elnet.predict(x_test_shape))
```

```
#Best ElasticNet MSE: 62.56902435370069, lambda = 0.19753166246833653
#alpha = 0.5
```

c)

In [ ]:
```
pc_reg = Pipeline([('pca', PCA()), ('ridge', Ridge())])
grid = {'pca__n_components':np.arange(2, 24, 2), 'ridge__alpha':[0.01, 0.05]+list(np.arange(0.1, 1, 10))}
pc_cv = GridSearchCV(pc_reg, grid, scoring='neg_mean_squared_error', cv=10, refit=True)
pc_cv.fit(x_train_shape, y_train)
```

In [ ]:
```
best = pc_cv.best_estimator_
opt_n = pc_cv.best_params_['pca__n_components']
opt_lambda = pc_cv.best_params_['ridge__alpha']
pc_err = MSE(y_test, best.predict(x_test_shape))

# Best PCR MSE: 62.32194871839432, lambda = 0.05
```

d)

In [72]:
```
partial_ls = PLSRegression()
partial_lscv = GridSearchCV(partial_ls, grid={'n_components':np.arange(2, 21, 2)}, scoring='neg_mean_squared_error', cv=10)
partial_lscv.fit(x_train_shape, y_train)
best = partial_lscv.best_estimator_
```

In [ ]:
```
opt_n = partial_lscv.best_params_['n_components']
partial_lscv_err = MSE(y_test, best.predict(x_test_shape))

#Best PLS MSE: 63.927709350623815
```

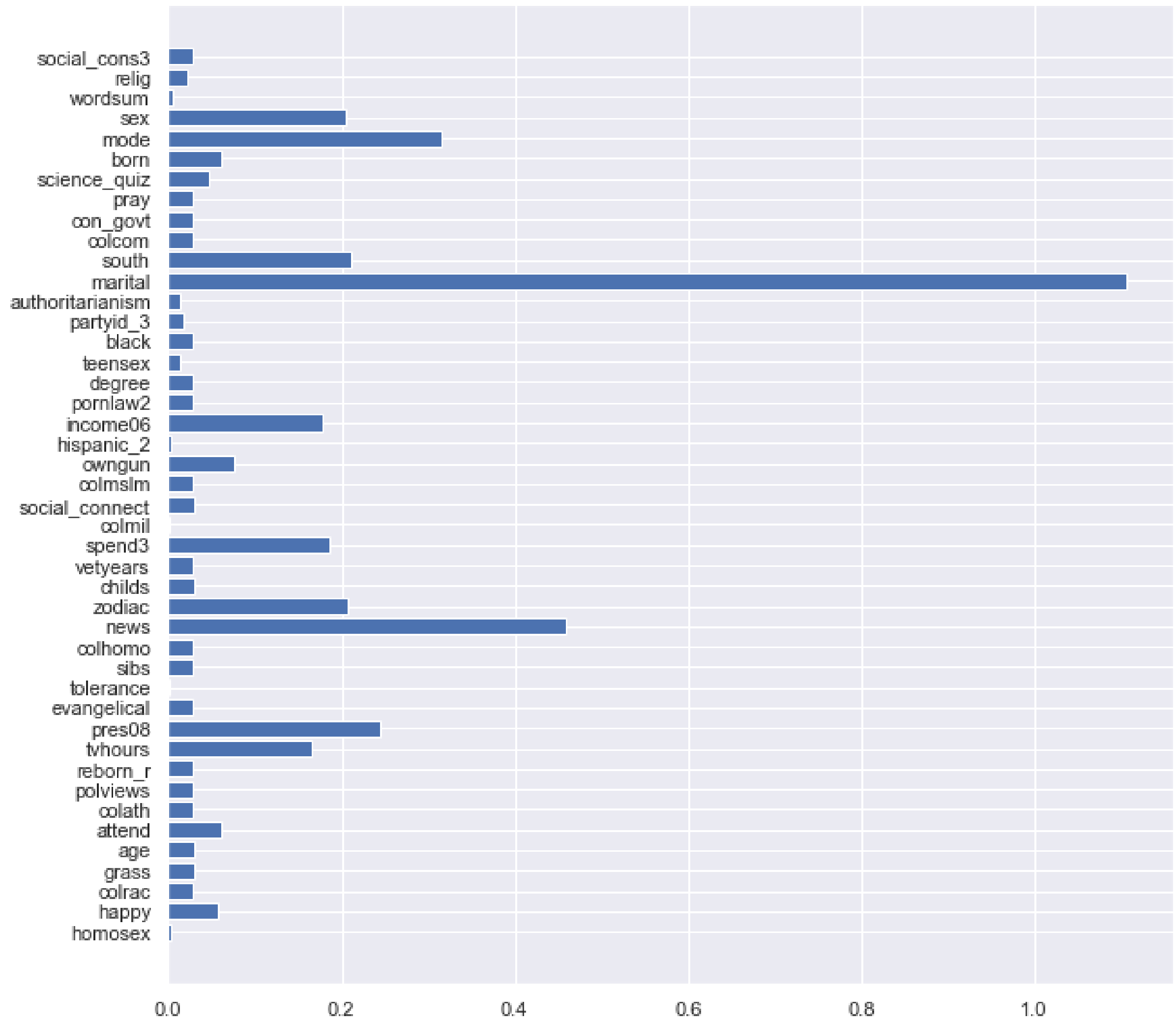
1. (20 points) For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other (e.g., talk about

feature importance, conditional effects, how these are ranked differently across different models, etc.).

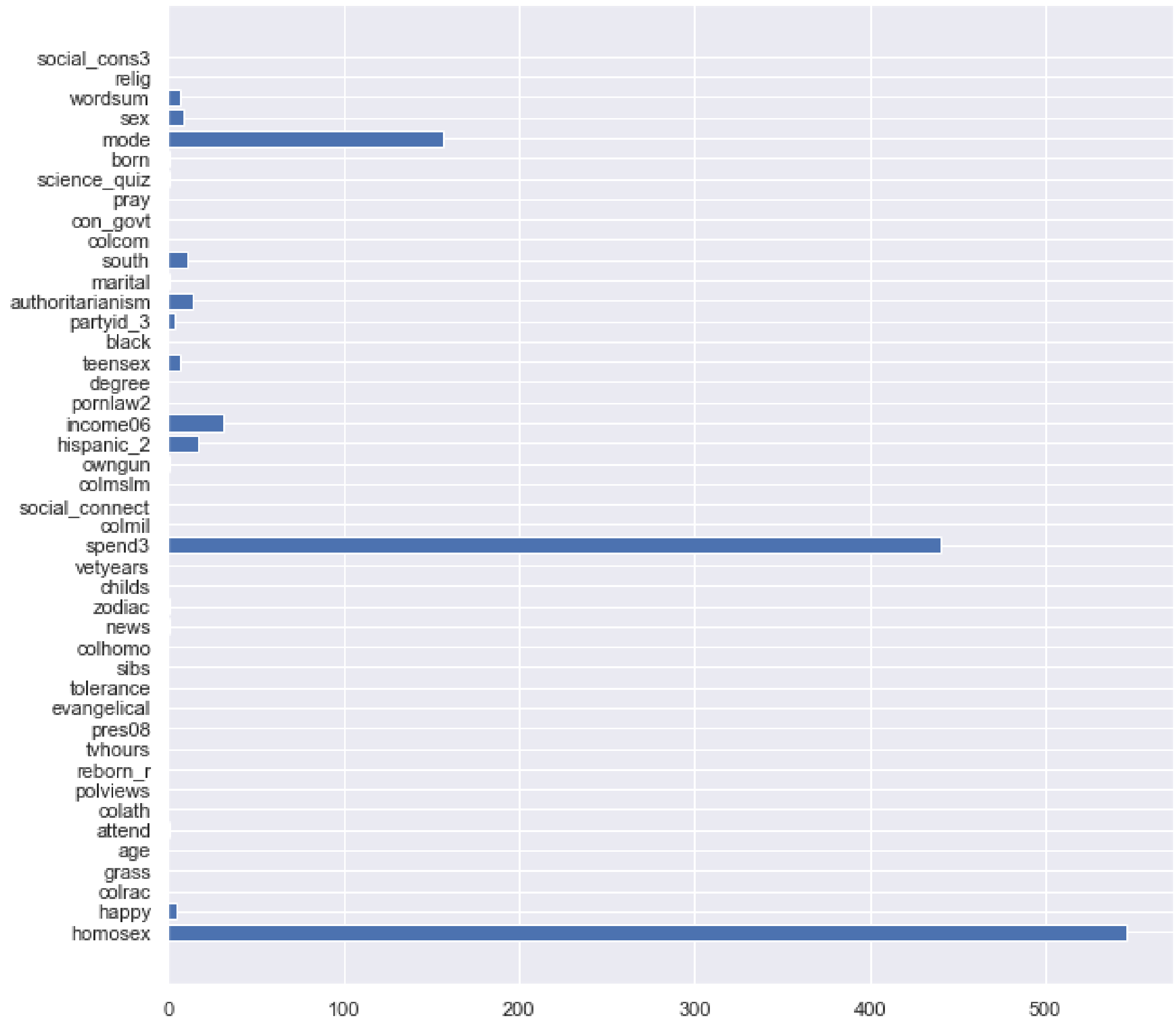
In [113]:
```
from scipy import stats

def feature_imp(mod, data, y, x_test, y_test, header):
    assert y in data.columns
    columns = set(data.columns)
    columnss.remove(y)
    yname, value = [], []
    for i, column in enumerate(columnss):
        if y == column: continue
        yname.append(column)
        x_amend = np.zeros(x_test.shape)
        x_amend[:, i] = x_test[:, i]
        y_amend = model.predict(x_amend)
        y_amend = y_amend.reshape(y_test.shape)
        value.append(stats.kruskal(y_amend, y_test)[0])
    plt.figure(figsize=(10, 10))
    plt.barh(np.arange(len(yname)), value)
    plt.yticks(np.arange(len(yname)), yname)
```

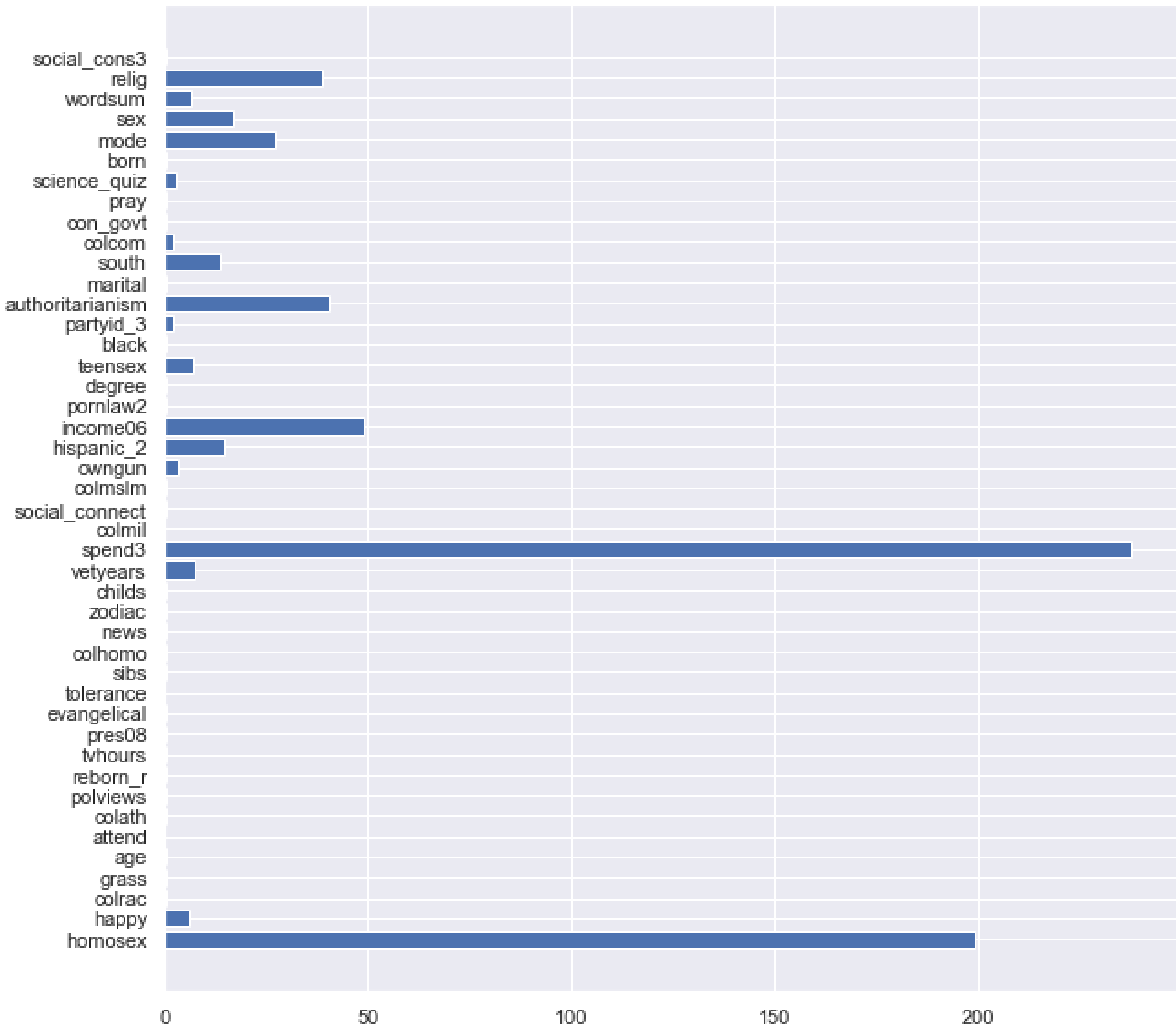
In [114]:
```
feature_imp(best_linreg, gss_tr, 'egalit_scale', x_test_shape, y_test,
'Linear Regression')
```



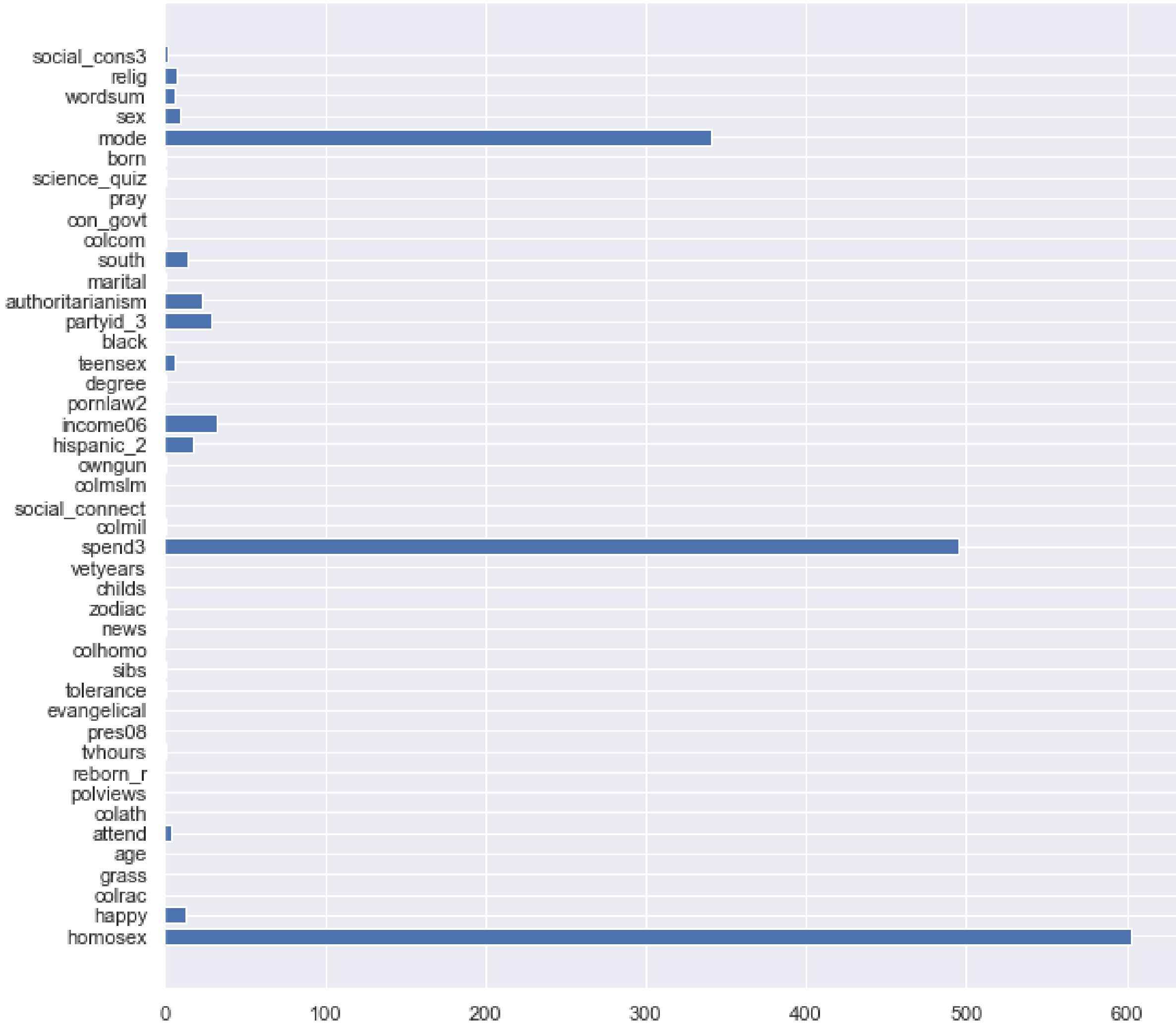
In [115]:
```
feature_imp(elnet, gss_tr, 'egalit_scale', x_test, y_test, 'Elastic Ne
t')
```



In [116]:
```
feature_int_plot(best_pca, gss_tr, 'egalit_scale', x_test, y_test, 'PC
R')
```



In [118]:
```
feature_imp(best_pls, gss_tr, 'egalit_scale', x_test, y_test, 'PLS')
```



- The different models used reach similar conclusions (regarding feature importance). We can see that 'homosex' and 'spend3' have a very high H score across all of the models (except linear regression), implying that they strongly predict egalitarianism scores. Interestingly, 'marital' is a strong predictor of egalitarianism in the linear regression, but not in any of the other models. Additionally, many of the features in the linear regression model seem to be interacting together to inform egalitarianism scores (though 'marital' is the most influential, followed by 'news'), while the other models only have a couple of predominant/influential predictors (for example, 'mode', 'spend3', and 'homosex' in partial least squares). The models besides linear regression seem to have done a better job of isolating the most relevant features that predict egalitarianism ('spend3', and 'homosex' across elastic net, pcr, and pls; 'mode' isn't quite as weighted through pcr).

In [ ]:

```

```