Deng_Yehong_HW4

February 16, 2020

Egalitarianism and income 1. (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.

[1]:
```
import pandas as pd
import numpy as np
import sklearn.linear_model as skl_lm
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
import statsmodels.api as sm
import seaborn as sns
from patsy import dmatrix
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
income_train = gss_train['income06']
income_test = gss_test['income06']
gss_train.head(10)
```

[1]:
```
   age         attend  authoritarianism black born  childs       colath  \
0   21          Never                 4    No  YES       0  NOT ALLOWED
1   42          Never                 4    No  YES       2      ALLOWED
2   70       <Once/yr                 1   Yes  YES       3      ALLOWED
3   35  Sev times/yr                 2    No  YES       2      ALLOWED
4   24  Sev times/yr                 6    No   NO       3  NOT ALLOWED
5   28          Never                 1   Yes  YES       2      ALLOWED
6   28        Once/yr                 1    No  YES       0      ALLOWED
7   55       Every wk                 2    No  YES       6      ALLOWED
8   36          Never                 1   Yes  YES       3      ALLOWED
9   28          Never                 4    No  YES       4      ALLOWED

        colrac     colcom       colmil  … social_connect social_cons3  \
```

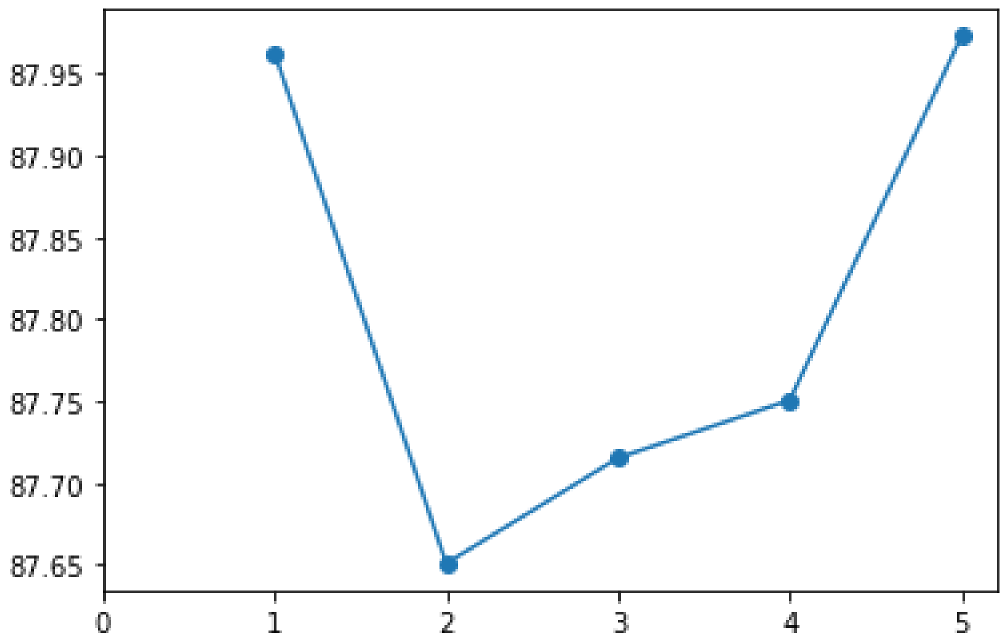
```
0  NOT ALLOWED      FIRED  NOT ALLOWED  …              5          Mod
1  NOT ALLOWED  NOT FIRED      ALLOWED  …              5      Liberal
2  NOT ALLOWED  NOT FIRED      ALLOWED  …              5      Liberal
3  NOT ALLOWED      FIRED  NOT ALLOWED  …             10      Liberal
4  NOT ALLOWED      FIRED      ALLOWED  …              4          Mod
5  NOT ALLOWED      FIRED  NOT ALLOWED  …              5      Liberal
6      ALLOWED  NOT FIRED      ALLOWED  …              7      Liberal
7      ALLOWED  NOT FIRED      ALLOWED  …              5          Mod
8  NOT ALLOWED  NOT FIRED  NOT ALLOWED  …              8      Liberal
9      ALLOWED  NOT FIRED      ALLOWED  …              4      Liberal

      south   spend3           teensex tolerance tvhours vetyears wordsum  \
0  Nonsouth  Conserv      ALWAYS WRONG        10       3     NONE       5
1  Nonsouth      Mod  NOT WRONG AT ALL        13       3     NONE       6
2  Nonsouth  Conserv      ALWAYS WRONG        10       3     NONE       6
3  Nonsouth  Liberal      ALWAYS WRONG        11       3     NONE       6
4  Nonsouth  Conserv  ALMST ALWAYS WRG         7       2     NONE       4
5  Nonsouth  Liberal   SOMETIMES WRONG         8       3     NONE       3
6  Nonsouth  Liberal      ALWAYS WRONG        14       3     NONE       7
7  Nonsouth      Mod      ALWAYS WRONG        11       6     NONE       2
8  Nonsouth  Liberal  NOT WRONG AT ALL        12       2     NONE       6
9  Nonsouth  Conserv   SOMETIMES WRONG        14       3     NONE       5

      zodiac
0      ARIES
1      ARIES
2     TAURUS
3    SCORPIO
4    SCORPIO
5      LIBRA
6     TAURUS
7     PISCES
8  CAPRICORN
9  CAPRICORN

[10 rows x 45 columns]
```

[2]:
```
#Referenced from http://www.science.smith.edu/~jcrouser/SDS293/labs/lab7-py.html
lm = skl_lm.LinearRegression()
poly_train = gss_train['income06'].values.reshape(-1,1)
poly_test = gss_test['income06'].values.reshape(-1,1)
crossvalidation = KFold(n_splits = 10, random_state = 1, shuffle =False)
mse_poly = []
for i in range(1,6):
    poly = PolynomialFeatures(degree=i)
    income_poly = poly.fit_transform(poly_train)
    model = lm.fit(income_poly, y_train)
```

```
    scores = cross_val_score(model, income_poly, y_train, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
    mse_poly.append(np.mean(np.abs(scores)))
print(mse_poly)
plt.plot(np.arange(1,6), mse_poly, marker = 'o')
plt.xticks(np.arange(0,6))
```

```
[87.96300193115192, 87.65085217286715, 87.71542380949792, 87.75064121251346,
87.97322944685062]
```

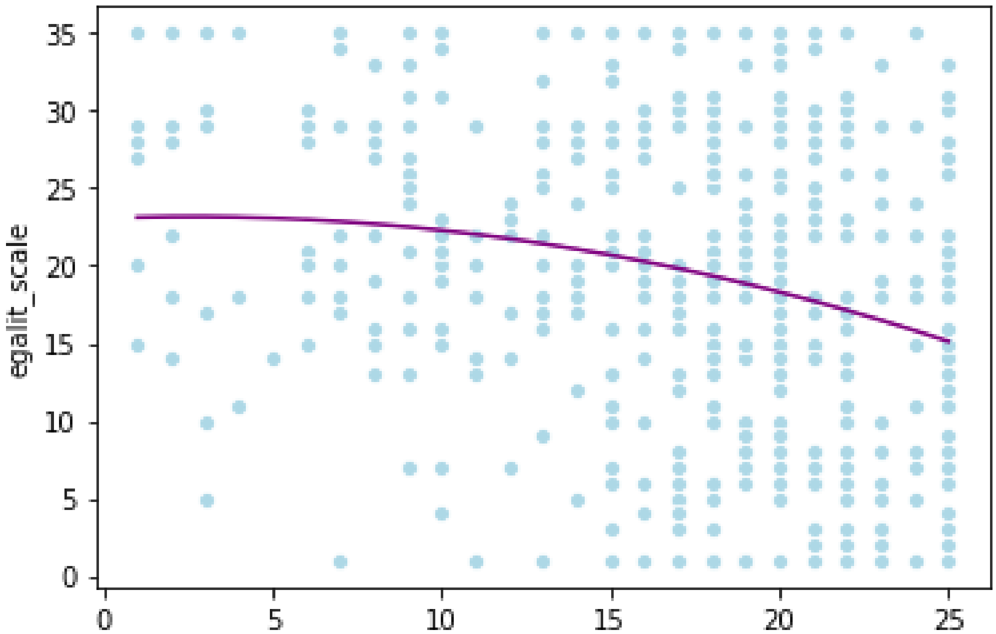
[2]:
```
([<matplotlib.axis.XTick at 0x1e0d0385748>,
  <matplotlib.axis.XTick at 0x1e0d0375ec8>,
  <matplotlib.axis.XTick at 0x1e0d03758c8>,
  <matplotlib.axis.XTick at 0x1e0d03bdc08>,
  <matplotlib.axis.XTick at 0x1e0d03bdac8>,
  <matplotlib.axis.XTick at 0x1e0d03c9a48>],
 <a list of 6 Text xticklabel objects>)
```



[3]:
```
#As the graph shown, the best degree with lowest MSE is 2
best_poly = PolynomialFeatures(2).fit_transform(poly_train)
fit_poly = lm.fit(best_poly, y_train)

income_best_test = PolynomialFeatures(2).fit_transform(poly_test)
poly_pred = fit_poly.predict(income_best_test)

sns.scatterplot(income_test.ravel(), y_test, color = "lightblue")
```

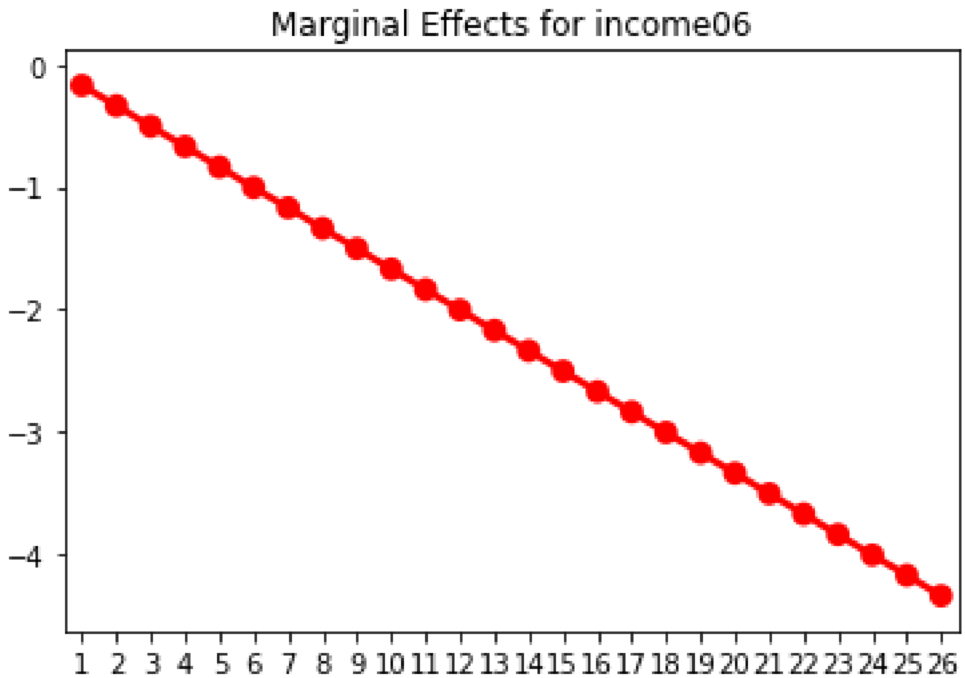
```
sns.lineplot(income_test.ravel(), poly_pred, color = 'purple')

print(income_train.min(),income_train.max() + 1)
```

```
1 26
```



[4]:
```
coef = fit_poly.coef_
def cal_marginal_effect(x):
    eff = 0
    for i in range(2):
        eff += coef[i] * (i+1) * (x**(i))
    return eff
marginal_effect = []
for income in range (1,27):
    marginal_effect.append(-cal_marginal_effect(income))
sns.pointplot(np.arange(1,27), marginal_effect, color = 'red').
 ↪set_title("Marginal Effects for income06")
```

[4]:
```
Text(0.5, 1.0, 'Marginal Effects for income06')
```



Both plots for fitted polynomial regression model at the optimal degree of 2 and the resulting marginal effect show that the income06 is negatively associated with the egalit scale. More specifically, our fitted polynomial model is a monotonicaly decreasing curvilinear prediction line, while the marginal effect plot is a straight down-ward sloping line.

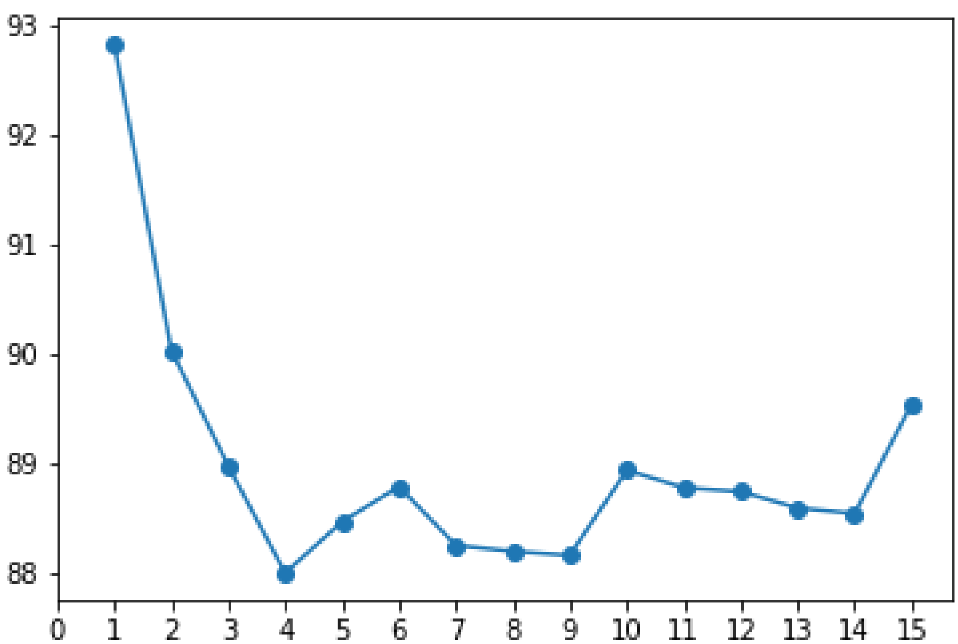
2. (20 points) Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.

[5]:
```
#step function referenced from http://www.science.smith.edu/~jcrouser/SDS293/
↪labs/lab12-py.html
import warnings
warnings.filterwarnings('ignore')
mse_step = []
for i in range(1,16):
    df_cut, bins = pd.cut(income_train, i, retbins = True, right = True)
    df_steps = pd.concat([income_train,df_cut,y_train], keys = ['income',␣
↪'income_cuts', 'egalit'], axis = 1)
    df_steps_dummies = pd.get_dummies(df_steps['income_cuts'])
    df_steps_dummies = sm.add_constant(df_steps_dummies)
    fit = lm.fit(df_steps_dummies, df_steps.egalit)
    scores = cross_val_score(fit, df_steps_dummies, df_steps.egalit, scoring =␣
↪"neg_mean_squared_error", cv = crossvalidation)
    mse_step.append(np.mean(np.abs(scores)))
print(mse_step)
```

```
plt.plot(np.arange(1,16), mse_step, marker = 'o')
plt.xticks(np.arange(0,16))
```

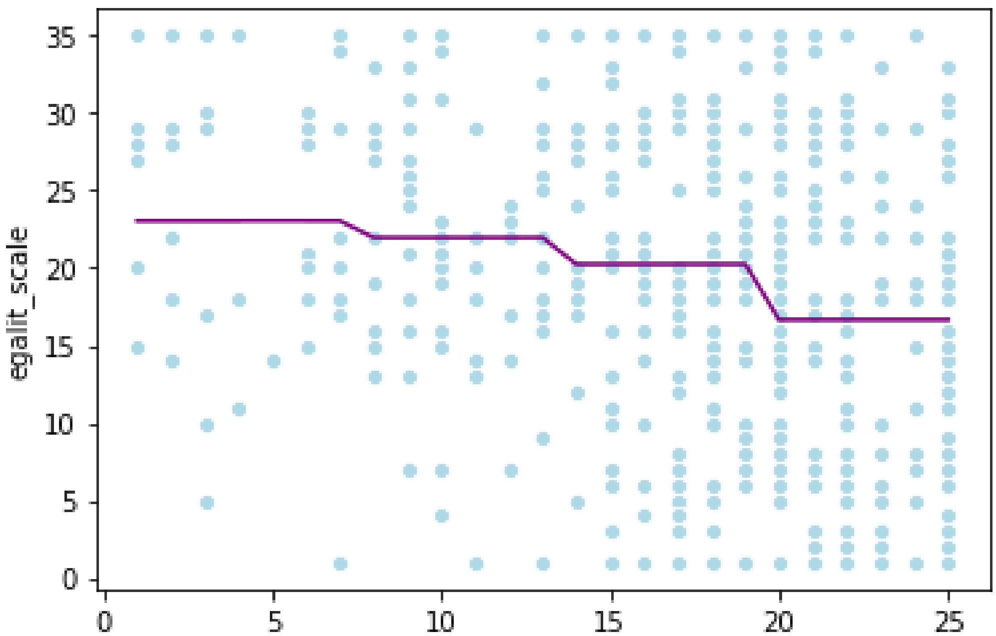
```
[92.8207554704396, 90.02781799497726, 88.97028485125078, 87.99268202181436,
88.46142270485102, 88.78295714948203, 88.24549422096784, 88.19032257110334,
88.15367930073336, 88.9332027781478, 88.7729266797524, 88.73680110797348,
88.58770878783058, 88.53586865677164, 89.53147235672544]
```

[5]:
```
([<matplotlib.axis.XTick at 0x1e0d0865ac8>,
  <matplotlib.axis.XTick at 0x1e0d0850d88>,
  <matplotlib.axis.XTick at 0x1e0d085fec8>,
  <matplotlib.axis.XTick at 0x1e0d0890388>,
  <matplotlib.axis.XTick at 0x1e0d0890a08>,
  <matplotlib.axis.XTick at 0x1e0d0893288>,
  <matplotlib.axis.XTick at 0x1e0d0893c08>,
  <matplotlib.axis.XTick at 0x1e0d0893cc8>,
  <matplotlib.axis.XTick at 0x1e0d08988c8>,
  <matplotlib.axis.XTick at 0x1e0d089b248>,
  <matplotlib.axis.XTick at 0x1e0d089bdc8>,
  <matplotlib.axis.XTick at 0x1e0d08a28c8>,
  <matplotlib.axis.XTick at 0x1e0d08a03c8>,
  <matplotlib.axis.XTick at 0x1e0d08a0e88>,
  <matplotlib.axis.XTick at 0x1e0d08a7988>,
  <matplotlib.axis.XTick at 0x1e0d08ab488>],
 <a list of 16 Text xticklabel objects>)
```



[6]:
```
#as shown from the plot, the best cut is 4
best_cut, best_bins = pd.cut(income_train, 4, retbins = True, right = True)
best_steps_dummies = pd.get_dummies(best_cut)
best_steps_dummies = sm.add_constant(best_steps_dummies)
step_fit = lm.fit(best_steps_dummies, y_train)

bin_mapping = np.digitize(income_test.ravel(), best_bins, right = True)
step_test = sm.add_constant(pd.get_dummies(bin_mapping))
step_pred = step_fit.predict(step_test)

sns.scatterplot(income_test.ravel(), y_test, color = "lightblue")
sns.lineplot(income_test.ravel(), step_pred, color = 'purple')
```

[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1e0d08e4608>
```



As we can see from the graph that shows the egalit scale at the 4 bins of income06, there is a general negative association between the egalit scale and the income06.When income06 increases, the egalit scale decreases.

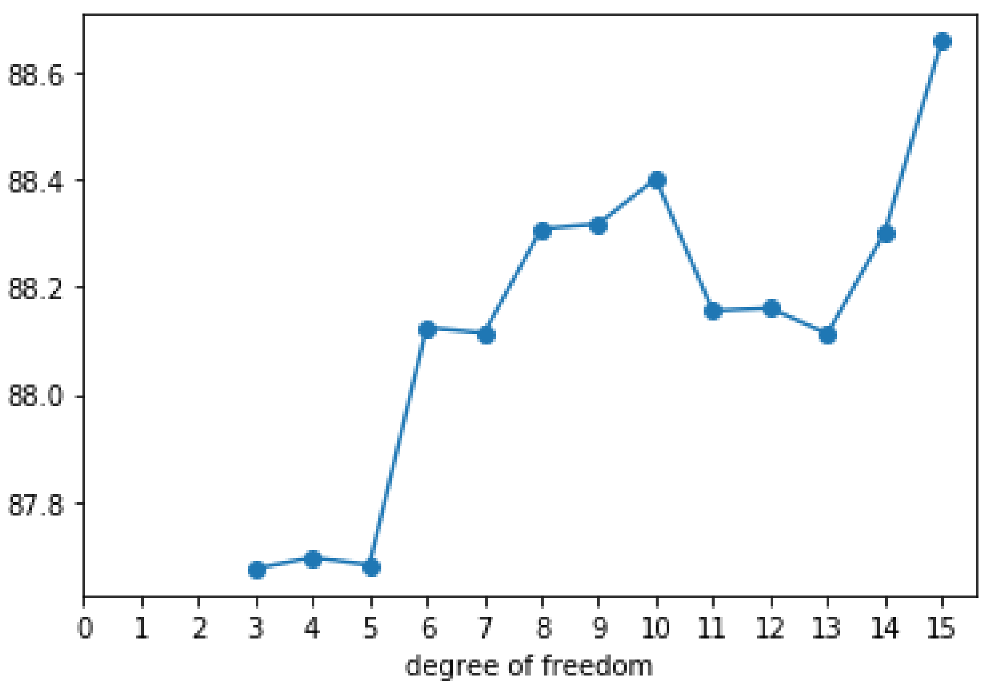
3. (20 points) Fit a natural regression spline to predict egalit_scale as a function of income06. Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the results of the optimal model.

[7]:
```
#referenced from http://www.science.smith.edu/~jcrouser/SDS293/labs/lab13-py.
↪html
import statsmodels.formula.api as smf
```

```
mse_ns = []
for i in range(3, 16):
    trans_income = dmatrix("cr(income_train, df = i)",
                           {'income_train': income_train},
                           return_type = 'dataframe')
    ns_model = lm.fit(trans_income, y_train)
    scores = cross_val_score(ns_model, trans_income, y_train, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
    mse_ns.append(np.mean(np.abs(scores)))
print(mse_ns)
plt.plot(np.arange(3,16), mse_ns, marker = 'o')
plt.xticks(np.arange(0,16))
plt.xlabel('degree of freedom')
```

```
[87.67428872142884, 87.69518211104575, 87.68123460571837, 88.12416716091863,
88.11412531161935, 88.30856828390921, 88.31787621764023, 88.40153862427135,
88.15514580946035, 88.1603854405639, 88.11249781002151, 88.30158574296163,
88.65904757765607]
```

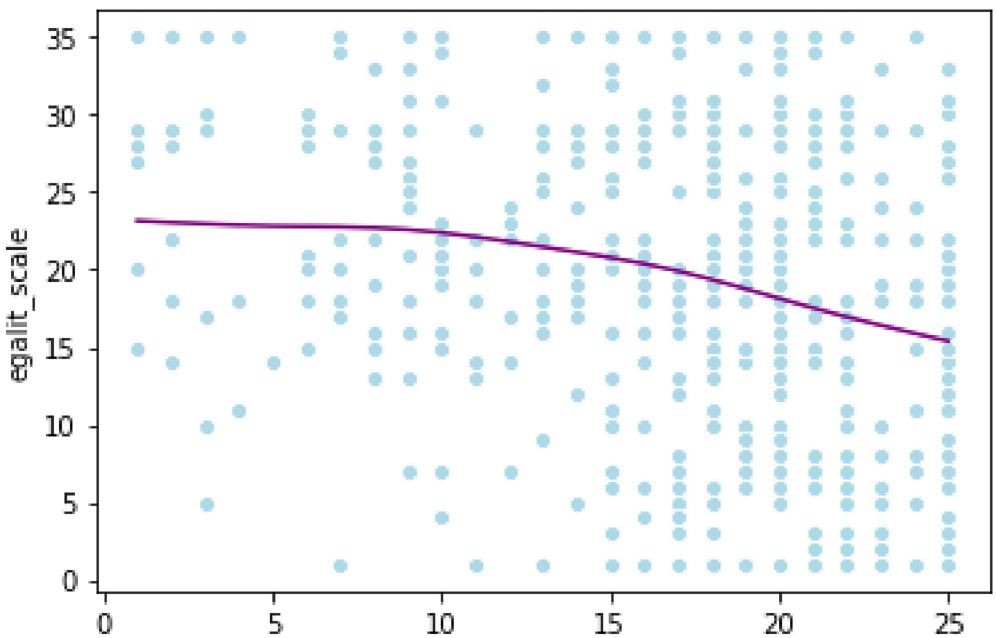
[7]:
```
Text(0.5, 0, 'degree of freedom')
```



[8]:
```
#As we can see in the list as well as in the plot, the best degree of freedom␣
 ↪is 3
trans_income = dmatrix("cr(income_train, df = 7)",
                       {'income_train': income_train},
```

```
                              return_type = 'dataframe')
ns_model = lm.fit(trans_income, y_train)

trans_test = dmatrix("cr(income_test, df = 7)",
                              {'income_test': income_test},
                              return_type = 'dataframe')
ns_pred = ns_model.predict(trans_test)

sns.scatterplot(income_test.ravel(), y_test, color = "lightblue")
sns.lineplot(income_test.ravel(), ns_pred, color = 'purple')
```

[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1e0d0a39d88>



As we can see in the mse plot, the best degree of freedom for natural regression spline is 3. Similar to the polynomial and step function, the plot of the natural regression spline also demonstrates a negative association between income06 and egalit scale.

Egalitarianism and everything 4. (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):

[9]:
```
#referenced from https://scikit-learn.org/stable/modules/preprocessing.html
from sklearn import preprocessing
x_train = gss_train.drop(['egalit_scale'], axis = 1)
x_test = gss_test.drop(['egalit_scale'], axis = 1)
```

```
scaler = preprocessing.StandardScaler()
def scaling(df):
    for i in df:
        if df[i].dtypes == object:
            df[i] = pd.get_dummies(df[i])
        elif df[i].dtypes == 'int64':
            reshape_data = df[i].values.reshape(-1,1)
            scaler.fit(reshape_data)
            df[i] = scaler.transform(reshape_data)
    return df
x_train_scaled = scaling(x_train)
x_test_scaled = scaling(x_test)

y_train_reshape = y_train.values.reshape(-1,1)
scaler.fit(y_train_reshape)
y_train_scaled = scaler.transform(y_train_reshape)

y_test_reshape = y_test.values.reshape(-1,1)
scaler.fit(y_test_reshape)
y_test_scaled = scaler.transform(y_test_reshape)
```

a. (5 points) Linear regression

[10]:
```
lm_model = lm.fit(x_train_scaled, y_train_scaled)
scores = cross_val_score(lm_model, x_train_scaled, y_train_scaled, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
lm_mse = np.mean(np.abs(scores))
print("The train MSE for the elastic net regression model is", lm_mse)
```

```
The train MSE for the elastic net regression model is 0.6970848933971507
```

b. (5 points) Elastic net regression

[18]:
```
from sklearn.linear_model import ElasticNetCV
from sklearn.metrics import mean_squared_error as mse
warnings.filterwarnings('ignore')
elastic_net = ElasticNetCV(l1_ratio = np.linspace(0.1,1,10), alphas = np.
 ↪linspace(0,0.01,11), cv = 10).fit(x_train_scaled, y_train_scaled)
MSE = mse(elastic_net.predict(x_train_scaled), y_train_scaled)
print("The train MSE for the elastic net regression model is", MSE)
```

```
The train MSE for the elastic net regression model is 0.6562766327047834
```

c. (5 points) Principal component regression

[23]:
```
#referenced from http://www.science.smith.edu/~jcrouser/SDS293/labs/lab11-py.
 ↪html
from sklearn.decomposition import PCA
#see the train data frame, it has 43 available predictors
```

```
min_mse = 100
min_comp = 0
for i in range(1,44):
    pca = PCA(n_components = i)
    x_reduced = pca.fit_transform(x_train_scaled)
    regr = lm
    scores = cross_val_score(regr, x_reduced[:,:i], y_train_scaled, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
    pca_mse = np.mean(np.abs(scores))
    if pca_mse <= min_mse:
        min_comp = i
        min_mse = pca_mse

print("The best components to select for the pca model is",min_comp)
```

```
The best components to select for the pca model is 24
```

[24]:
```
pca2 = PCA(n_components = 24)
x_reduced2 = pca2.fit_transform(x_train_scaled)
regr = lm
scores = cross_val_score(regr, x_reduced2[:,:24], y_train_scaled, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
pca_mse2 = np.mean(np.abs(scores))
print("The train MSE for pca is",pca_mse2)
```

```
The train MSE for pca is 0.683871408223884
```

d. (5 points) Partial least squares regression

[14]:
```
from sklearn.cross_decomposition import PLSRegression, PLSSVD
min_mse = 100
min_comp = 0
for i in range(1,44):
    pls = PLSRegression(n_components = i)
    scores = cross_val_score(pls, x_reduced[:,:i], y_train_scaled, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
    pls_mse = np.mean(np.abs(scores))
    if pls_mse <= min_mse:
        min_comp = i
        min_mse = pls_mse

print("The best components to select for the pls model is",min_comp)
```

```
The best components to select for the pls model is 30
```

[15]:
```
pls2 = PLSRegression(n_components = 30)
scores = cross_val_score(pls2, x_reduced[:,:30], y_train_scaled, scoring =␣
 ↪"neg_mean_squared_error", cv = crossvalidation)
```

```
pls_mse2 = np.mean(np.abs(scores))
print("The train MSE for pca is",pls_mse2)
```

```
The train MSE for pca is 0.680838523007422
```

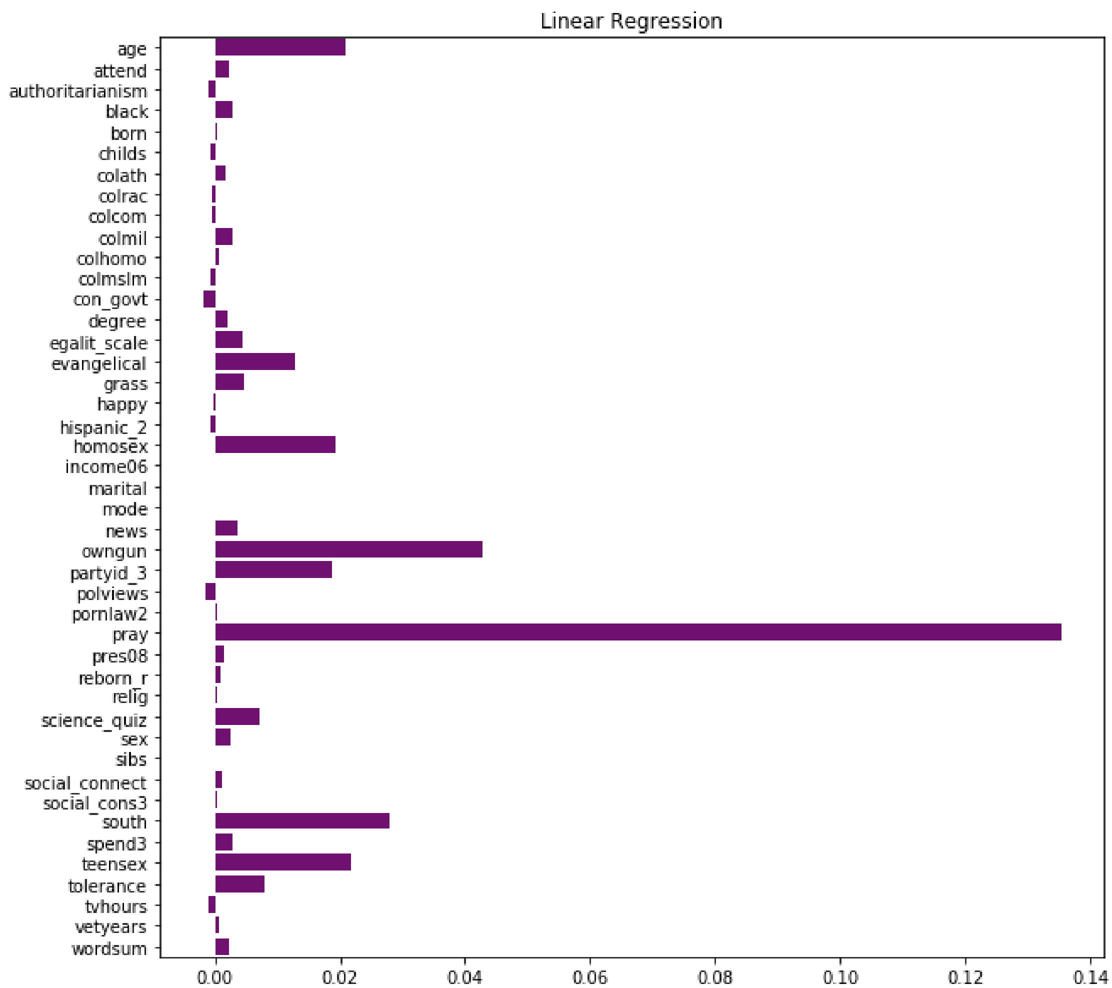
5. (20 points) For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other (e.g., talk about feature importance, conditional effects, how these are ranked differently across different models, etc.).

[39]:
```
#referenced from https://scikit-learn.org/stable/modules/generated/sklearn.
↪impute.SimpleImputer.html
#referenced from http://rasbt.github.io/mlxtend/user_guide/evaluate/
↪feature_importance_permutation/#example-1-feature-importance-for-regressors
from mlxtend.evaluate import feature_importance_permutation
from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(missing_values = np.nan, strategy = 'mean', verbose =␣
↪0)
imp_mean = imp_mean.fit(x_test_scaled)
imputed_test = imp_mean.transform(x_test_scaled)
```

[41]:
```
def plot_imp(model):
    imp_vals, _ = feature_importance_permutation(
        predict_method = model.predict,
        X = imputed_test,
        y = y_test_scaled,
        metric='r2',
        num_rounds = 10)


    col = []
    imp = []
    for i in range(x_test_scaled.shape[1]):
        col.append(gss_test.columns[i])
        imp.append(imp_vals[i])
    res = sns.barplot(x = imp, y =col, color = 'purple')

    return res
```

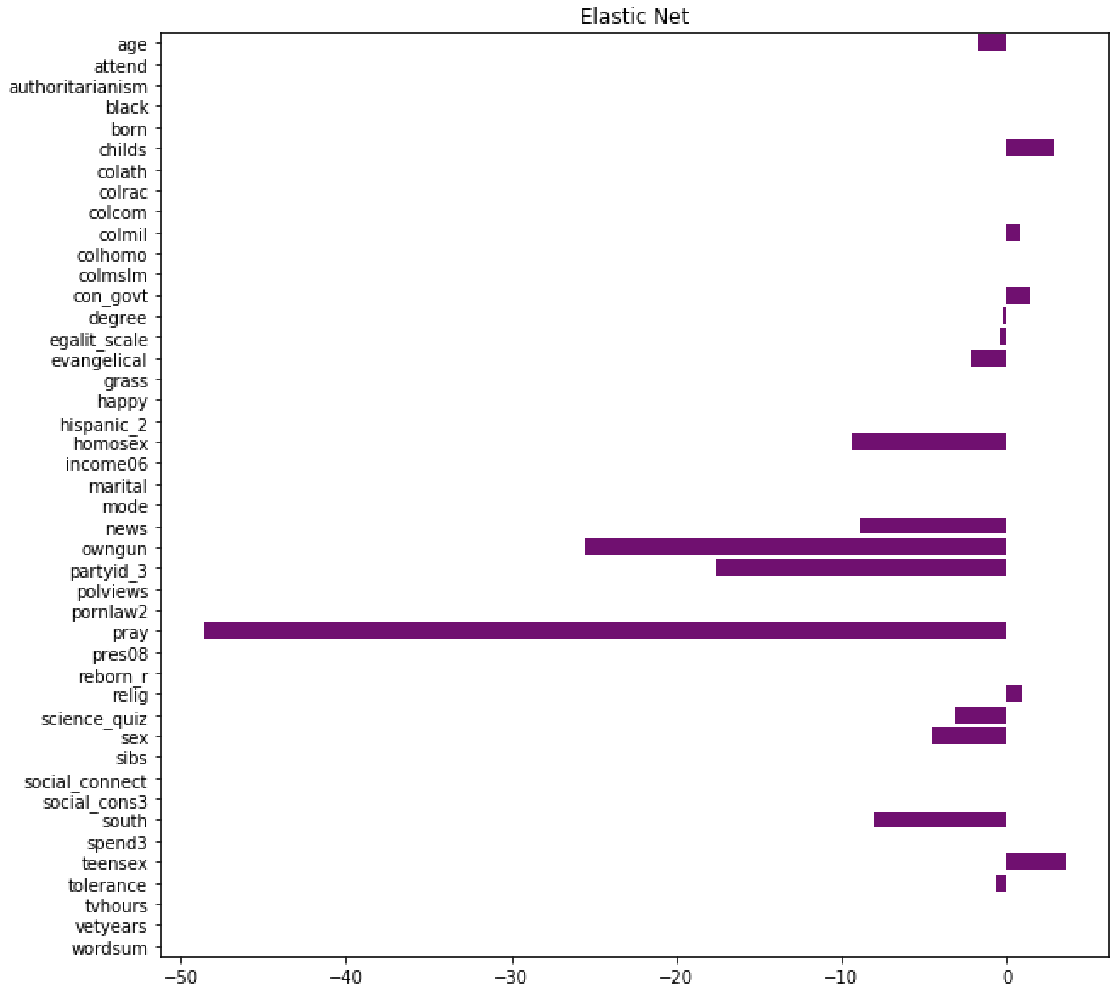
[45]:
```
plt.figure(figsize = (10,10))
lm = skl_lm.LinearRegression().fit(x_train_scaled, y_train_scaled)
lm_plot = plot_imp(lm)
plt.title('Linear Regression')
```

[45]:
```
Text(0.5, 1.0, 'Linear Regression')
```



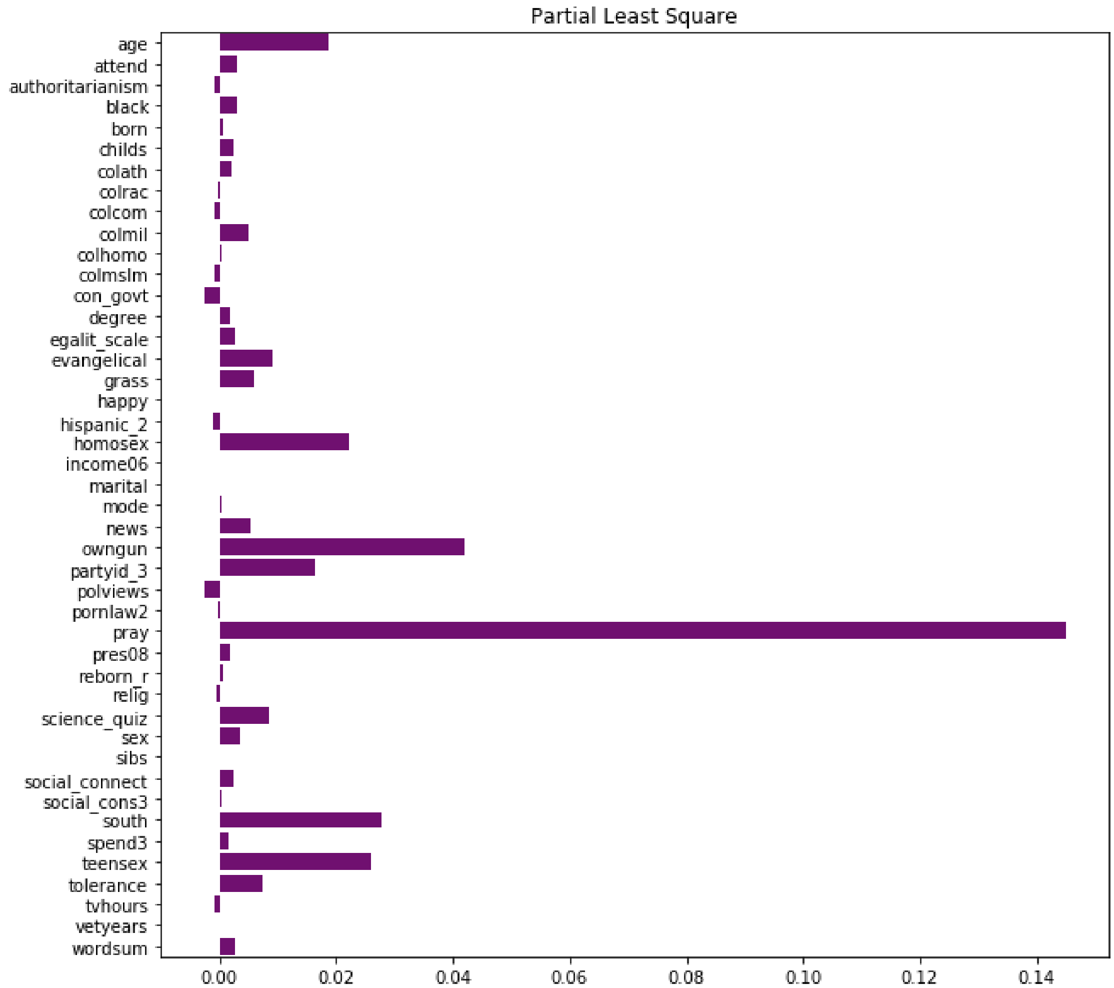
[46]:
```
plt.figure(figsize = (10,10))
lm_plot = plot_imp(elastic_net)
plt.title('Elastic Net')
```

[46]: Text(0.5, 1.0, 'Elastic Net')




```
[58]: pls_reg = pls2.fit(x_train_scaled, y_train_scaled)
      plt.figure(figsize = (10,10))
      lm_plot = plot_imp(pls_reg)
      plt.title('Partial Least Square')
```

```
[58]: Text(0.5, 1.0, 'Partial Least Square')
```



The graphs above are the feature importance plots for different models.As we can see from these plots, different predictors take differnt level of importances within different methods. For the Linear Regression model, the predictors that explain most of the variance are “pray”, “owngun”, “south”, and “age”. For the Elastic Net, the features that explain most of the variance are “teensex”, “child”, and ”“con_govt”. Nonetheless, the “pray” and “owngun” become the least important ones. For the PLS, the feature that explain most variance change back to “pray”, “owngun”,and “south”, which is the same as the Linear Regression. This happens since different models take different features into calculation and penalizing others at the same time.