HW4

In [10]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression, ElasticNetCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
import itertools
import statsmodels.api as sm
from mlxtend.feature_selection import SequentialFeatureSelector
as sfs
from patsy import dmatrix
from sklearn.decomposition import PCA
from sklearn.preprocessing import PolynomialFeatures, scale, Min
MaxScaler
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import KFold, cross_val_score
from mlxtend.evaluate import feature_importance_permutation
from sklearn.impute import SimpleImputer
from sklearn.inspection import plot_partial_dependence, partial_
dependence
```

```
Egalitarianism and income (Q1-Q3)
```

Q1.(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 [11]:

```
gss_train = pd.read_csv('data/gss_train.csv')
gss_test = pd.read_csv('data/gss_test.csv')
gss_train.head()
```

Out[11]:

	age	attend	authoritarianism	black	born	childs	colath	c
0	21	Never	4	No	YES	0	NOT ALLOWED	ALLC
1	42	Never	4	No	YES	2	ALLOWED	ALLC
2	70	<Once/yr	1	Yes	YES	3	ALLOWED	ALLC
3	35	Sev times/yr	2	No	YES	2	ALLOWED	ALLC
4	24	Sev times/yr	6	No	NO	3	NOT ALLOWED	ALLC

5 rows × 45 columns

In [12]:

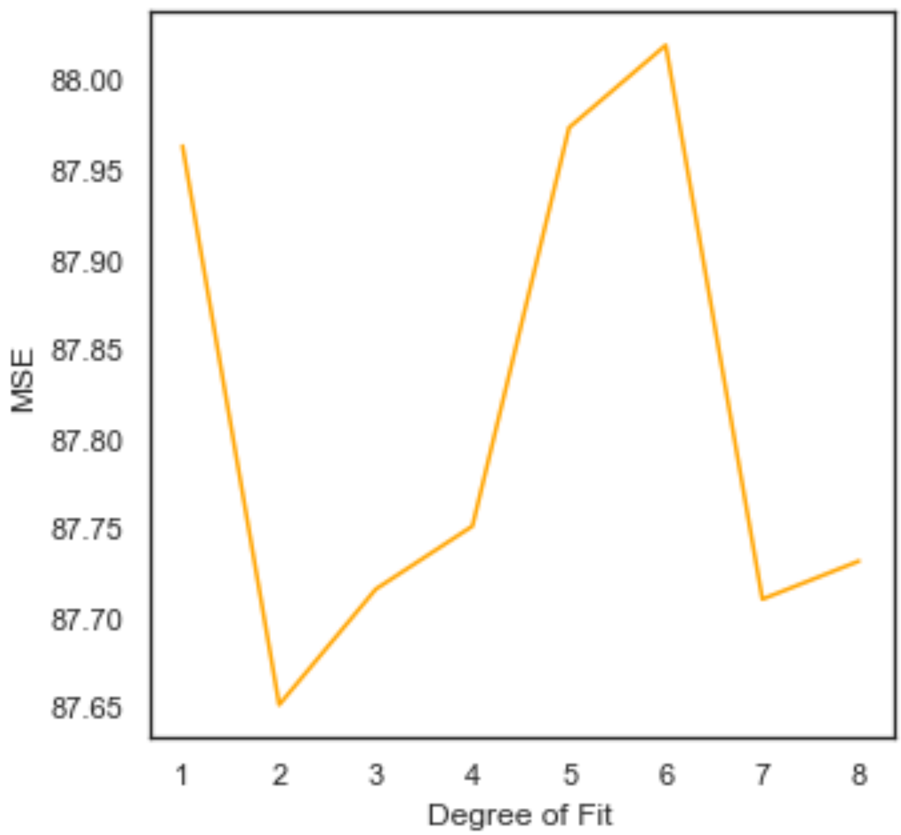
```
X_train = gss_train['income06'].values.reshape(-1,1)
X_test = gss_test['income06'].values.reshape(-1,1)
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
```

In [13]:

```
cv = KFold(n_splits=10, random_state=1, shuffle=False)
lm = LinearRegression()
cv_dict = {}
for i in range(1,9):
    pol = PolynomialFeatures(degree=i)
    X_pol = pol.fit_transform(X_train)
    model = lm.fit(X_pol, y_train)
    scores = cross_val_score(model, X_pol,
                                   y_train,
        scoring="neg_mean_squared_error", cv=cv)
    cv_dict[i] = np.mean(np.abs(scores))
```

In [14]:

```
sns.set(rc={'figure.figsize':(5,5)})
sns.set_style("white")
degree = range(1, 9)
plt.plot(degree, list(cv_dict.values()), color = 'orange')
plt.xlabel('Degree of Fit')
plt.ylabel('MSE')
plt.show()
```



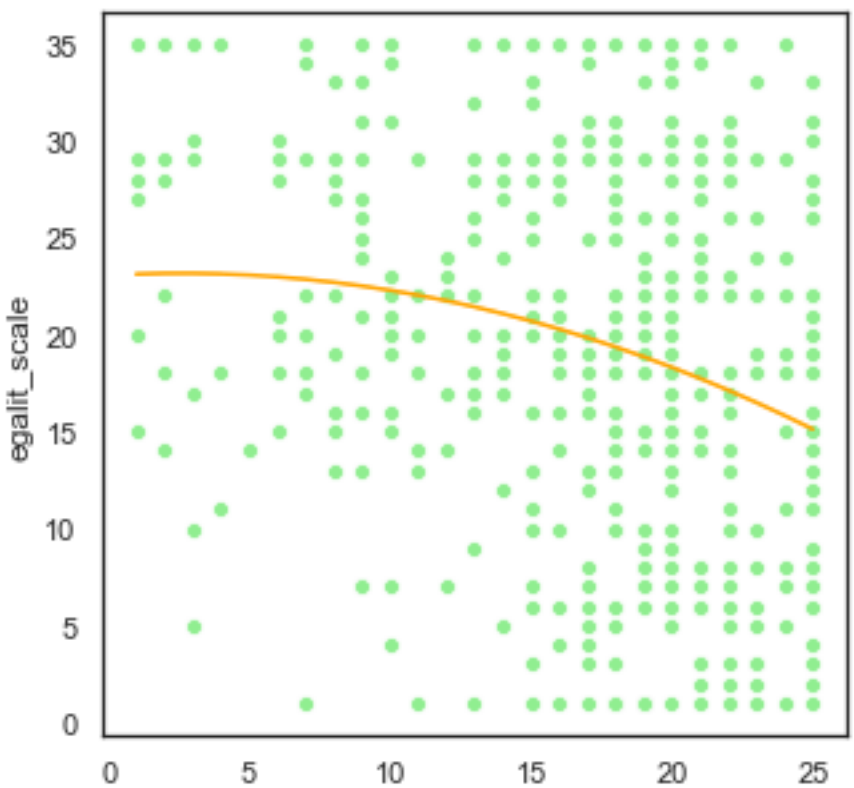
From the graph, we see that the optimal degree is 2.

In [15]:

```
poly = PolynomialFeatures(degree=2)
X_pol = poly.fit_transform(X_train)
model = lm.fit(X_pol, y_train)
sns.scatterplot(X_test.ravel(), y_test,
                color='lightgreen')
sns.lineplot(X_test.ravel(),
             model.predict(
                 poly.fit_transform(X_test)),
             color ='orange')
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c266efd
d0>
```



In [24]:

```
def get_ame (X, y, degree):
    x = pd.DataFrame()
    x_drv = pd.DataFrame()
    ame_total = []
    for deg in range(degree+1):
        if deg != 0:
            x[deg] = X ** deg
    lm = LinearRegression().fit(x, y)
    coefs = lm.coef_

    for deg in range(degree):
        x_drv[deg] = (deg+1)* coefs[deg] * (X ** deg)
        ame_x = x_drv[deg].mean()
        ame_total.append(ame_x)
    ame = np.sum(ame_total)
    return ame, coefs
```

In [25]:

```
ame, coefs = get_ame(gss_train['income06'],
                     gss_train['egalit_scale'], 10)
print('The AME of income is {}'.format(ame))
```

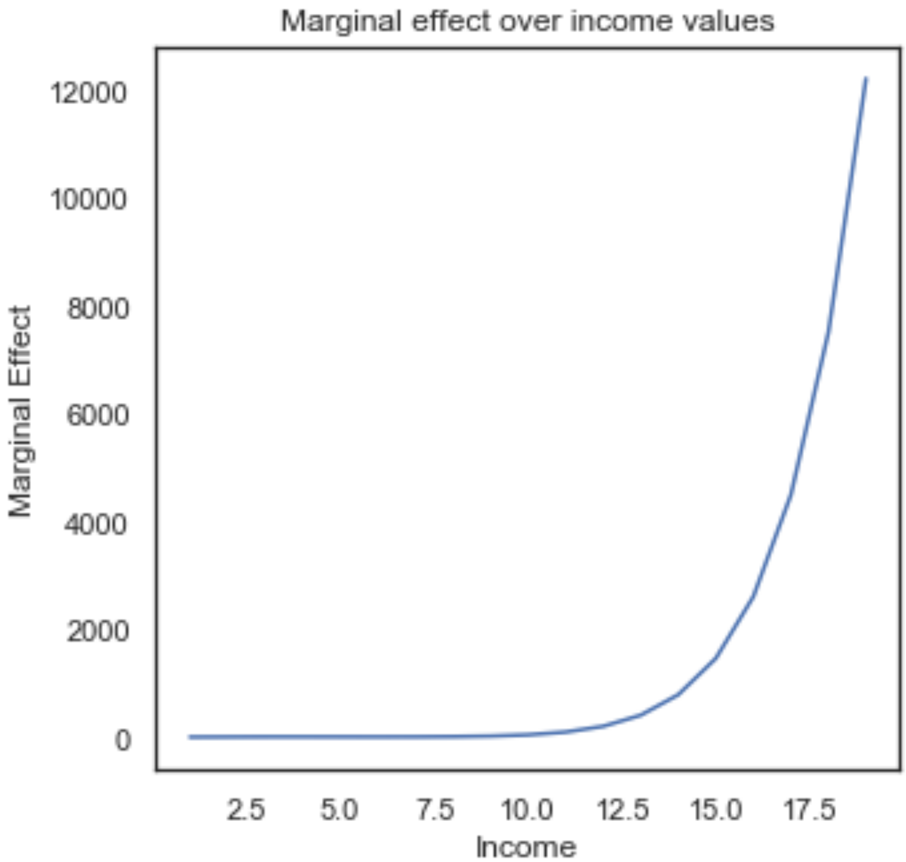
```
The AME of income is -1.762934161026351
```

In [26]:

```
def marginal_effect(x):
    y = 0
    for i in range(9):
        y += coefs[i] * (i+1) * (x**(i))
    return y
margin = []
for x in range(1,20):
    margin.append(marginal_effect(x))
```

In [27]:

```
income = np.arange(1,26)
marginal_income = pd.DataFrame(zip(list(income), margin),
columns = ['income_value','marginal_effect'])
plt.plot(marginal_income['income_value'],
         marginal_income['marginal_effect'])
plt.title('Marginal effect over income values')
plt.xlabel('Income')
plt.ylabel('Marginal Effect');
```



The graph above shows an overall negative correlation between marginal effect and income. Also, the marginal effect are negative.

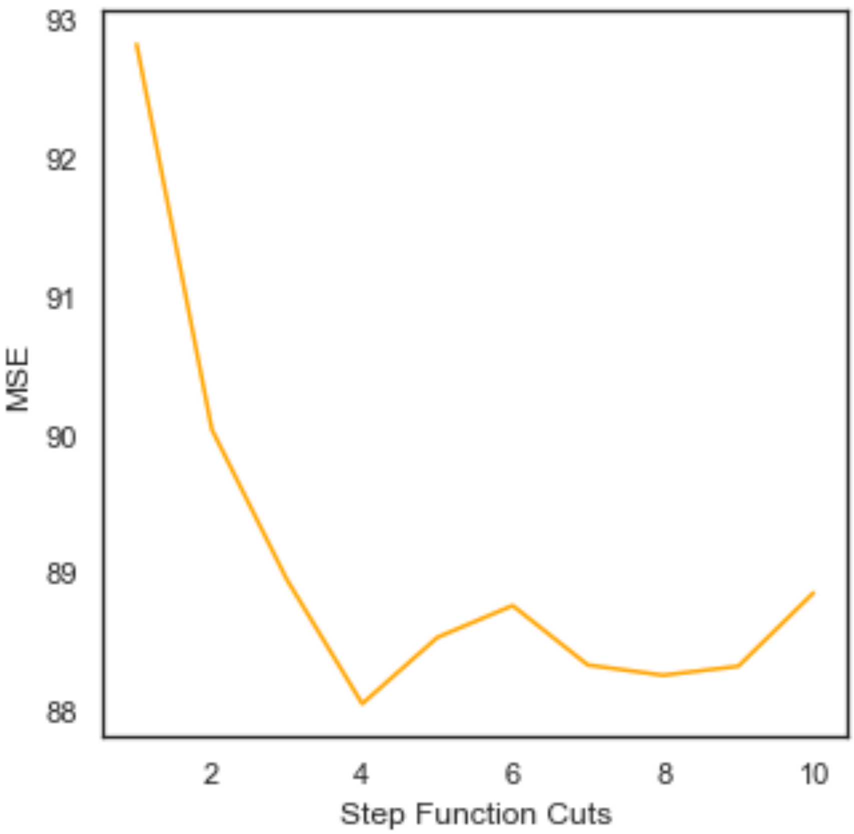
Q2.(20 points) Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.

In [28]:

```
sf_dict = {}
for i in range(1,11):
    df_cut, bins = pd.cut(X_train.ravel(), i,
                          retbins = True, right = True)
    dummies = pd.get_dummies(df_cut)
    dummies = sm.add_constant(dummies)
    model = lm.fit(dummies, y_train)
    scores = cross_val_score(model,
                             dummies, y_train,
                             scoring="neg_mean_squared_error",
                             cv=crossvalidation)
    sf_dict[i] = np.mean(np.abs(scores))
```

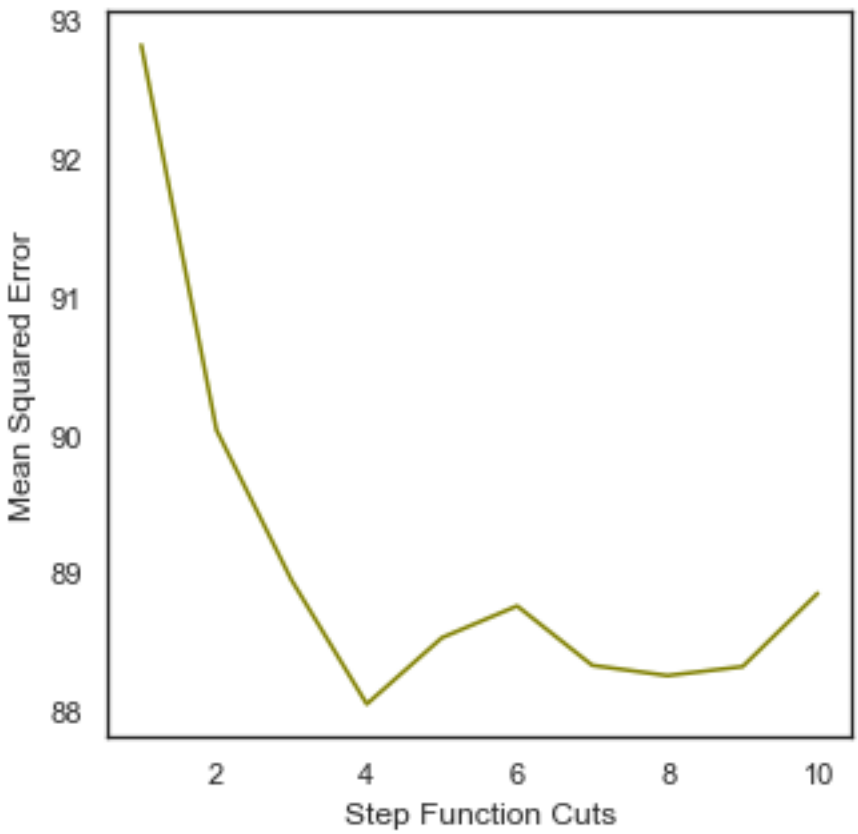
In [29]:

```
mg = range(1, 11)
plt.plot(mg, list(sf_dict.values()),
         color = 'orange')
plt.xlabel('Step Function Cuts')
plt.ylabel('MSE')
plt.show()
```



In [30]:

```
mg = range(1, 11)
plt.plot(mg, list(step_dict.values()), color = 'olive')
plt.xlabel('Step Function Cuts')
plt.ylabel('Mean Squared Error')
plt.show()
```



From the graph, we see that the optimal degree is 4.

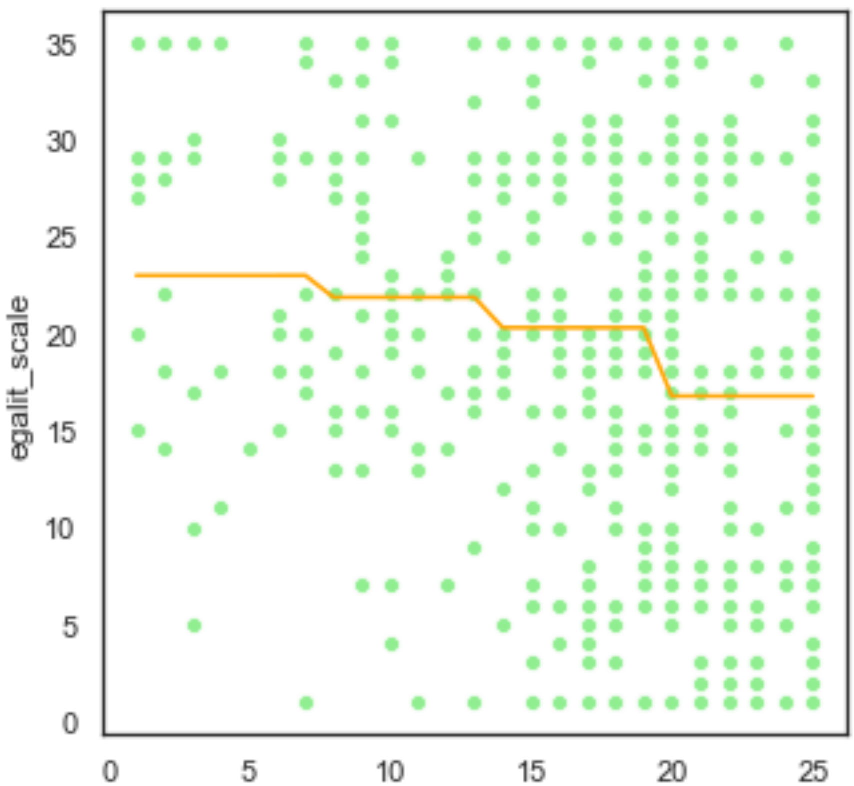
The plot above shows that the optimal number of bins is 4, which means that when the income is divided into four groups, we have the lowest error if we use the linear regression model to predict egalit_scale.

In [31]:

```
df_cut, bins = pd.cut(X_train.ravel(), 4,
                      retbins = True, right = True)
dummies = pd.get_dummies(df_cut)
dummies = sm.add_constant(df_dummies)
model = lm.fit(dummies, y_train)
bin_mapping = np.digitize(X_test.ravel(),
                          bins, right = True)
test_dummies = pd.get_dummies(bin_mapping)
test_dummies = sm.add_constant(test_dummies)
sns.scatterplot(X_test.ravel(), y_test,
                color='lightgreen')
sns.lineplot(X_test.ravel(),
             model.predict(test_dummies), color = 'orange')
```

Out[31]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c26af9e
d0>
```



Q3.(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 [32]:

```
spline_dict = {}
for i in range(3,11):
    trans_X = dmatrix("bs(train, df=i, degree=3)",
                        {"train": X_train,"df": i},
                                  return_type='dataframe')
model = lm.fit(trans_X, y_train)
scores = cross_val_score(model, trans_X, y_train,
                         scoring="neg_mean_squared_error",
                         cv=cv)
spline_dict[i] = np.mean(np.abs(scores))
print('The optimal degree of freedom is',
      max(spline_dict, key=spline_dict.get))
```

```
The optimal degree of freedom is 10
```

In [33]:

```
trans_X = dmatrix("bs(train, df=7, degree=3)",
                  {"train": X_train},
                  return_type='dataframe')
model = lm.fit(trans_X, y_train)
test_spline = dmatrix("bs(test, df=7, degree=3)",
                      {"test": X_test},
                      return_type='dataframe')
sns.scatterplot(X_test.ravel(),
                y_test,
                color='lightblue')
sns.lineplot(X_test.ravel(),
             model.predict(test_spline),
             color = 'orange')
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c26bf15
50>
```

Just like the polynomial regression and the step function, the predicted value of egalit scale and income06 are negatively correlated.

```
Egalitarianism and everything (Q4-Q5)
```

Q4.(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).

In [34]:

```
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
X_train = gss_train.drop('egalit_scale', axis=1)
X_test = gss_test.drop('egalit_scale', axis=1)
```

In [35]:

```
scaler = MinMaxScaler(feature_range=(0, 1))
def scale(df):
    if isinstance(df, pd.DataFrame):
        for column in df:
            if df[column].dtypes == object:
                df[column] = pd.get_dummies(df[column])
            elif df[column].dtypes == 'int64':
                reshape_col = df[column].values.reshape(-1,1)
                scaler.fit(reshape_col)
                df[column] = scaler.transform(reshape_col)
            else:
                reshape_col = df.values.reshape(-1, 1)
                scaler.fit(reshape_col)
                df = scaler.transform(reshape_col)
    return df
```

In [36]:

```
X_train = scale(X_train)
X_test = scale(X_test)
y_train = scale(y_train)
y_test = scale(y_test)
```

In [37]:

```
#a.Linear Regression
lm = LinearRegression().fit(X_train, y_train)
scores = cross_val_score(model,
                         X_train, y_train,
                         scoring="neg_mean_squared_error", cv=10
)
lm_mse = np.mean(np.abs(scores))
print('the test MSE of least squares linear: ',
      lm_mse)
```

```
the test MSE of least squares linear:  64.4871855211
3122
```

In [39]:

```
#b.Elastic net
alpha = np.arange(0, 1.1, step=0.1)
Elastic = ElasticNetCV(
    cv=10, alphas=alpha
).fit(X_train, y_train)
Elastic_mse = mean_squared_error(
    Elastic.predict(X_test), y_test)
print('l1 ratio: ', Elastic.l1_ratio_)
print('alpha: ', Elastic.alpha_)
print('The test MSE of elastic net is',
      Elastic_mse)
```

```
l1 ratio:  0.5
alpha:  0.1
The test MSE of elastic net is 63.23427900534246
```

In [40]:

```
#c.Principal component regression
pcr_dict = {}
for i in np.arange(0.3, 1, 0.05):
    pca = PCA(i)
    Xr = pca.fit_transform(X_train)
    reg = LinearRegression().fit(Xr, y_train)
    scores = cross_val_score(reg,
                             Xr, y_train,
                             scoring="neg_mean_squared_error",
                             cv=10)
    pcr_mse = np.mean(np.abs(scores))
    pcr_dict[i] = pcr_mse
min(pcr_dict, key=pcr_dict.get)
```

Out[40]:

```
0.7
```

In [41]:

```
pca = PCA(0.7)
Xr = pca.fit_transform(X_train)
reg = LinearRegression().fit(Xr, y_train)
scores = cross_val_score(reg,
                         Xr, y_train,
                         scoring="neg_mean_squared_error",
                         cv=10)
pcr_mse = np.mean(np.abs(scores))
print("The test MSE of principal component regression is",
      pcr_mse)
```

```
The test MSE of principal component regression is 64
.32783247681823
```

In [42]:

```
#d.Partial least squares regression
pls_dict = {}
for i in np.arange(1, 45):
    pls = PLSRegression(i).fit(X_train, y_train)
    scores = cross_val_score(pls,
                             X_train, y_train,
                             scoring="neg_mean_squared_error",
                             cv=10)
    pls_mse = np.mean(np.abs(scores))
    pls_dict[i] = pls_mse
min(pls_dict, key=pls_dict.get)
```

Out[42]:

```
6
```

In [43]:

```
pls = PLSRegression(6).fit(X_train, y_train)
scores = cross_val_score(pls,
                         X_train, y_train,
                         scoring="neg_mean_squared_error", cv=10
)
pls_mse = np.mean(np.abs(scores))
print("The test MSE of partial least squares is",
      pls_mse)
```

```
The test MSE of partial least squares is 64.41523584
190291
```

Q5.(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.).

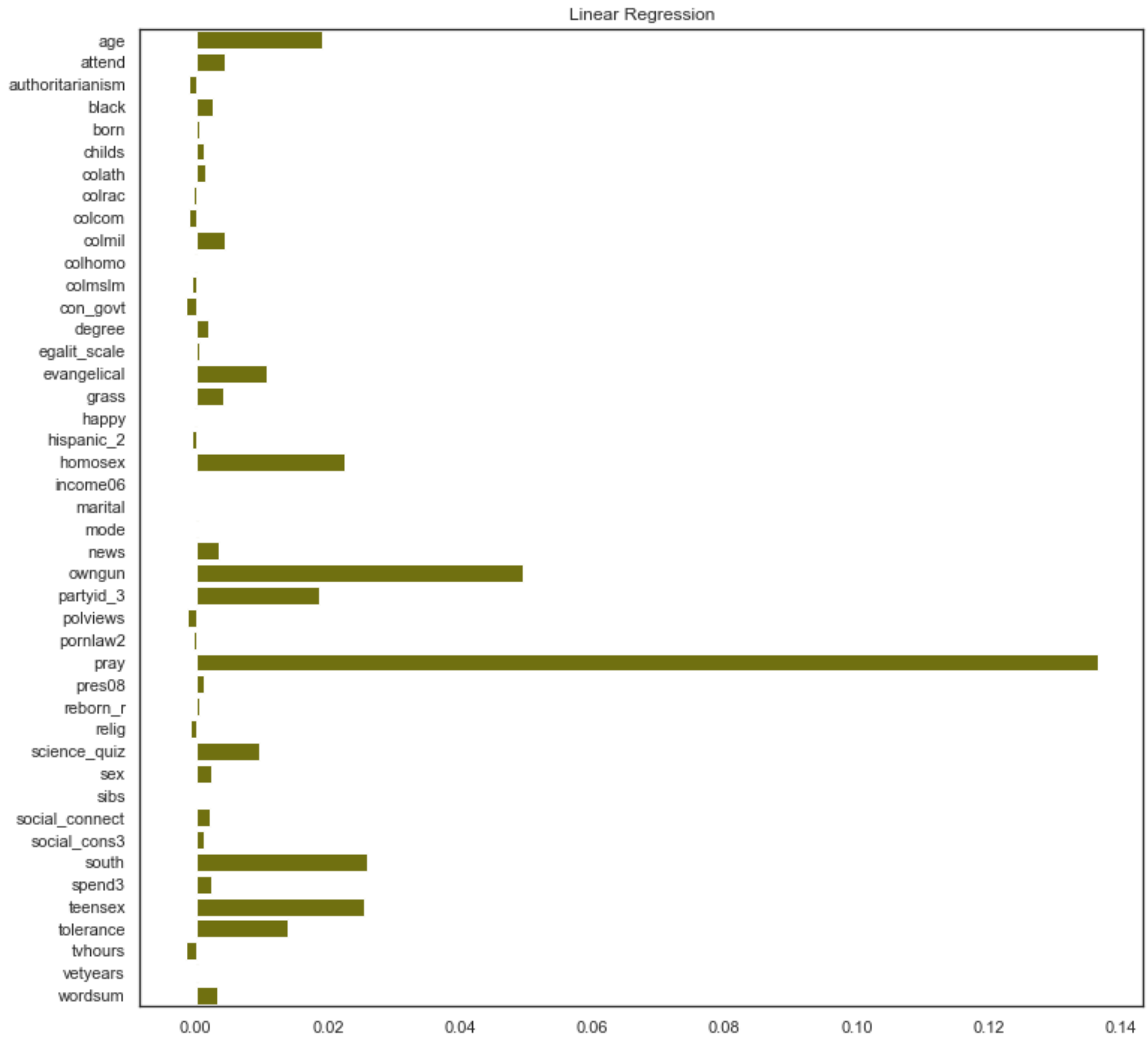
In [48]:

```
ipt = SimpleImputer(missing_values = np.nan,
                    strategy = 'mean', verbose=0)
ipt = ipt.fit(X_test)
impute_test = ipt.transform(X_test)

def plot_imp(model):
    imp_vals, _ = feature_importance_permutation(
        predict_method=model.predict,
        X=impute_test,y=y_test,
        metric='r2',num_rounds=10)
    col = []
    imp = []
    for i in range(X_test.shape[1]):
        col.append(gss_test.columns[i])
        imp.append(imp_vals[i])
    ax = sns.barplot(x=imp, y=col, color='olive')
    return ax
```

In [49]:

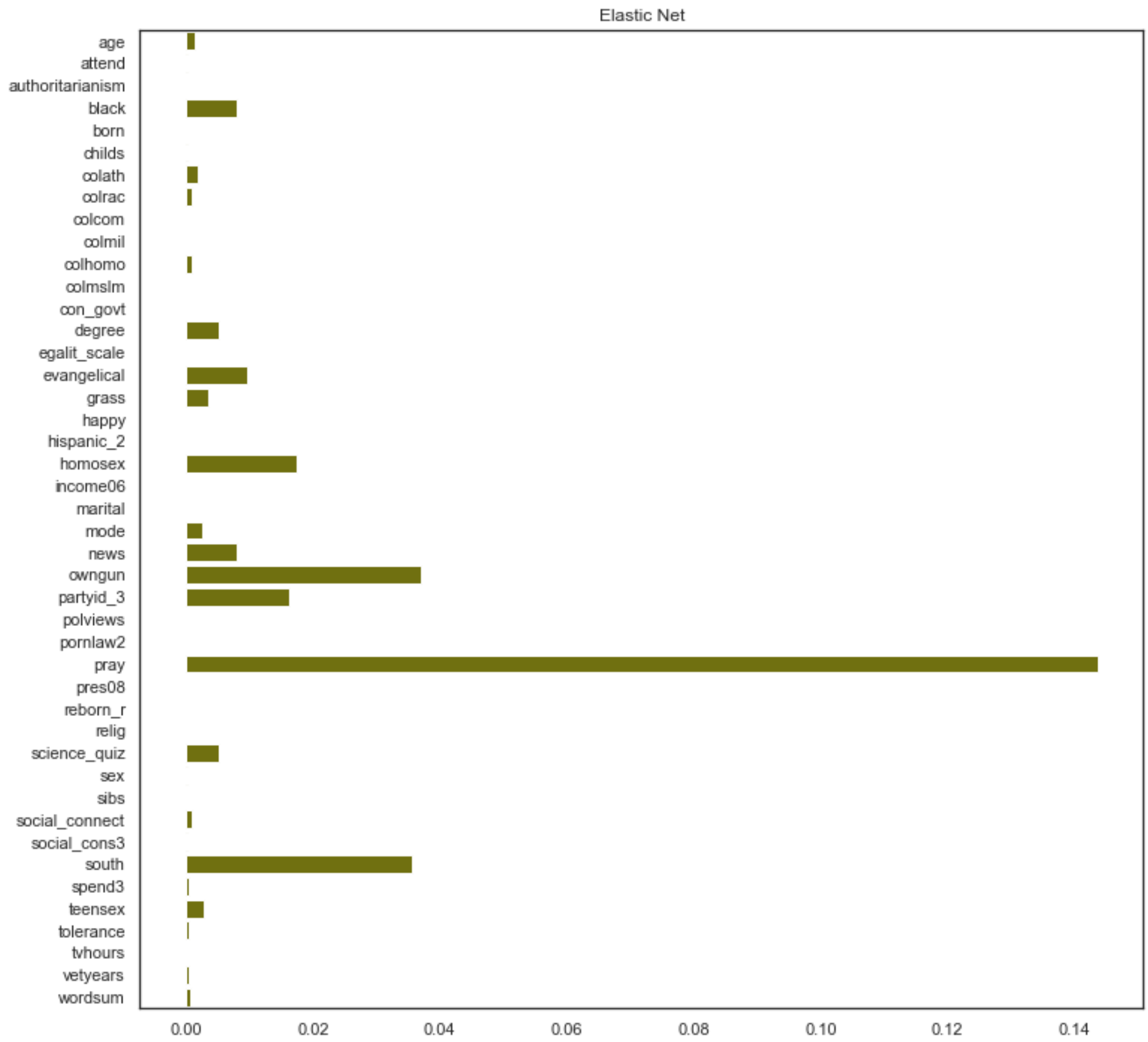
```
plt.figure(figsize=(12, 12))
lm_plot = plot_imp(lm)
plt.title('Linear Regression');
```



The Linear Regression graph tells us that “pray”,“owngun”, “age”, and “south” explain most of the variance.

In [50]:

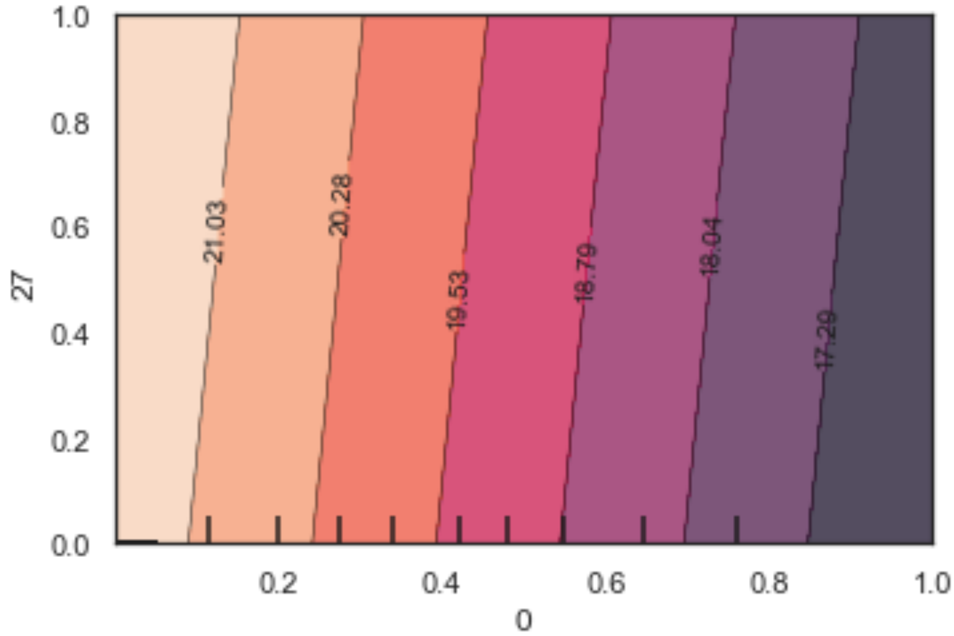
```
plt.figure(figsize=(12, 12))
elasticnet_plot = plot_imp(Elastic)
plt.title('Elastic Net');
```



The Elastic Net graph also demonstrates that 'pray' is of signicant importance among these variables.

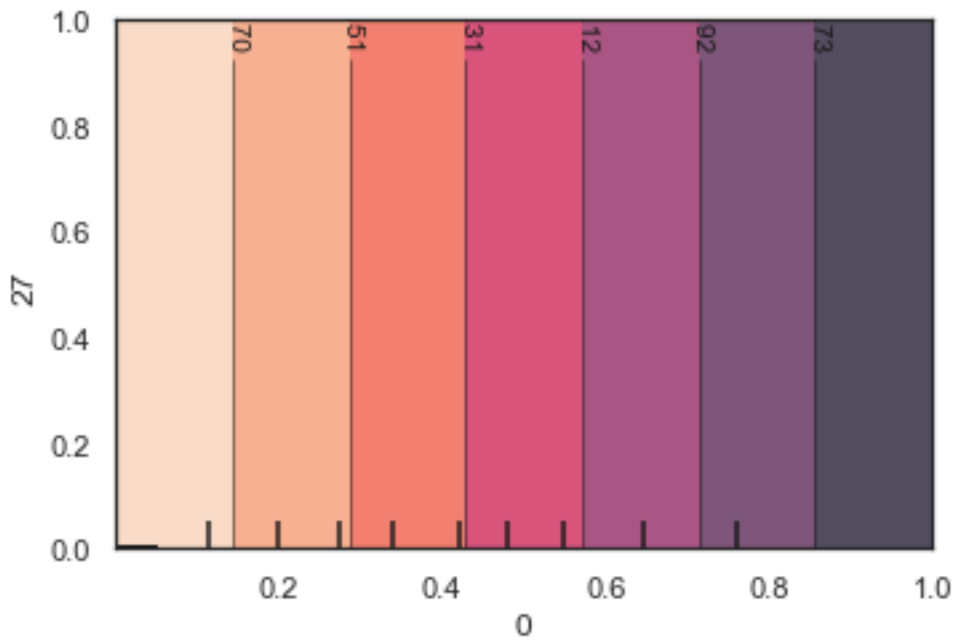
In [57]:

```
plot_partial_dependence(lm, X_test, [(0, 27)])
```



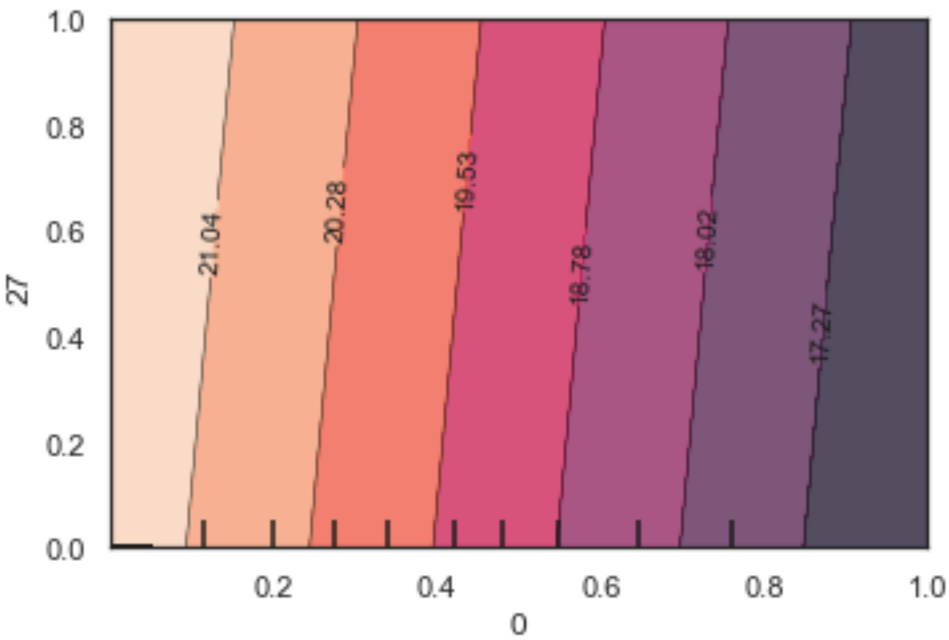
In [61]:

```
plot_partial_dependence(Elastic, X_test, [(0, 27)])
```



In [60]:

```
plot_partial_dependence(pls, X_test, [(0, 27)])
```



As the line graphs give information that 'pray' and 'age' are clearly two leading variables that explain most of the variance, we look at the partial dependence of these two and how they interact with each other.

Interestingly, we see different interaction patterns from these models. In linear regression and PLS regression, the interaction of the two seem to be linear, but in Elastic net regression, the variable 'pray' seems to be independent from 'age'.

From the plots, we can see that the interactions happened in very different ways. In Elastic net regression, “pray” is nearly independent of “age” at every level. This is consistent with their R2 values, in which both features did not explain well the variance in elastic model. The interactions in linear regression and PLS regression tend to be linear. This is consistent with our previous findings.

The reason of this discrepancy may result from the attribute of 'pray', that is, the change in feature may not have a large effect of 'pray'.

In [ ]: