In [107]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.api as sm
import sklearn
from patsy import dmatrix
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
from sklearn.base import BaseEstimator
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.linear_model import ElasticNetCV
from sklearn.inspection import plot_partial_dependence, partial_depend
ence
from statsmodels.tools.tools import add_constant
from mlxtend.evaluate import feature_importance_permutation
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings('ignore')
```

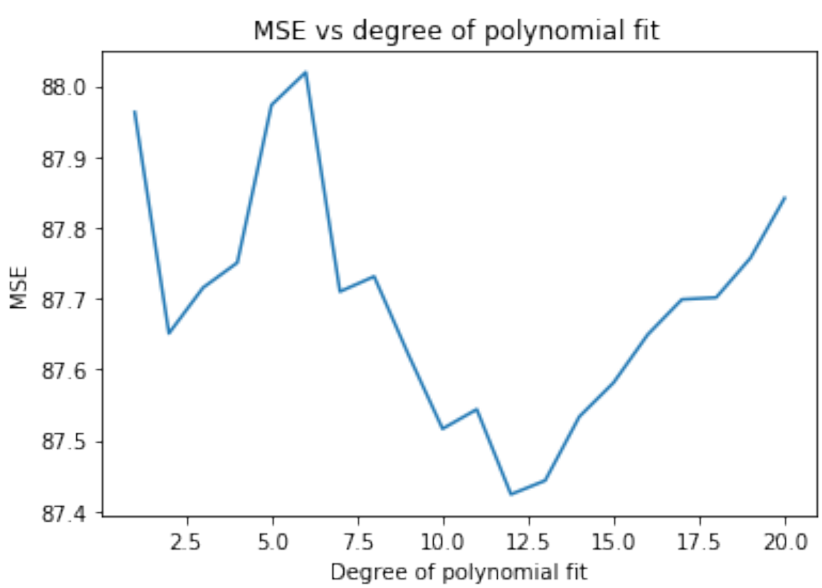
1.1 Perform polynomial regression to predict egalit_scale as a function of income06. Use and plot 10-fold cross-validation to select the optimal degree d for the polynomial based on the MSE. Plot the resulting polynomial fit to the data, and also graph the average marginal effect (AME) of income06 across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

In [5]:
```
gss_test = pd.read_csv("gss_test.csv")
gss_train = pd.read_csv("gss_train.csv")
gss_test.dropna(inplace=True)
gss_train.dropna(inplace=True)
```

In [47]:
```
np.random.seed(1234)
```

In [48]:
```
X = np.array(gss_train['income06'])
y = np.array(gss_train['egalit_scale'])
```

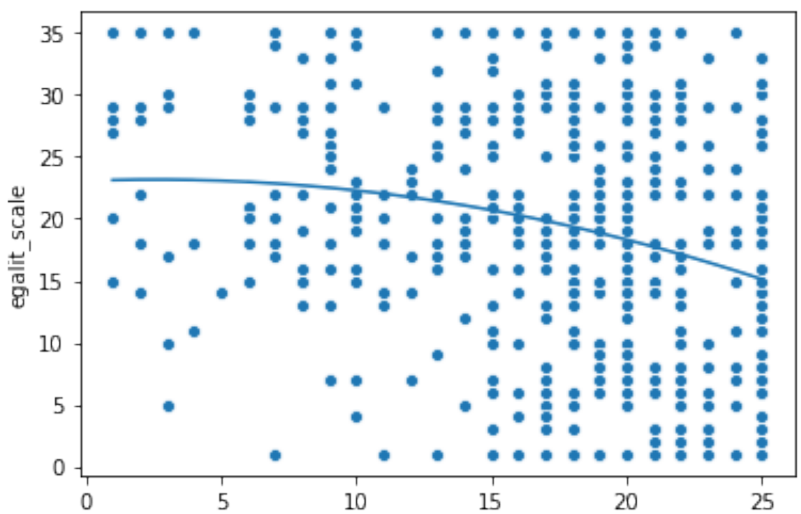
In [49]:
```
#Plot the MSE graph
def get_degree(X, y):
    x = pd.DataFrame()
    cv_dict = {}
    min_error = 10000000000
    min_degree = 0
    degrees = np.arange(1, 21)

    for d in degrees:
        x[d] = X ** d
        lr = LinearRegression()
        error = np.sum(-cross_val_score(lr, x, y, cv=10, scoring='neg_
mean_squared_error'))/10
        cv_dict[d] = error
    if error < min_error:
        min_error = error
        min_degress = d
    return min_degree, cv_dict

lists = sorted(get_degree(gss_train['income06'],y)[1].items())
degree, mse = zip(*lists)
plt.figure()
plt.plot(degree, mse)
plt.xlabel('Degree of polynomial fit')
plt.ylabel('MSE')
plt.title('MSE vs degree of polynomial fit ');
```



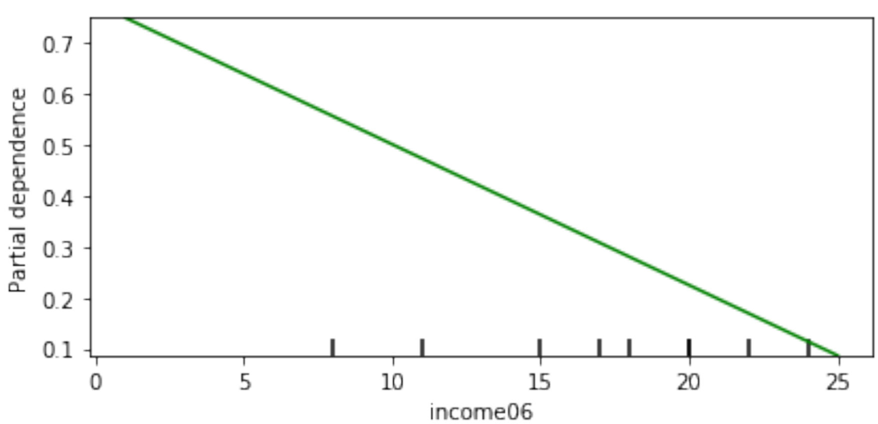
Through 10-fold cross validation, we find the optimal degree is approximately 12

In [60]:
```
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
X_train = gss_train['income06'].values.reshape(-1,1)
X_test = gss_test['income06'].values.reshape(-1,1)
lm = LinearRegression()
poly = PolynomialFeatures(degree=2)
X_poly = poly.fit_transform(X_train)
model = lm.fit(X_poly, y_train)
sns.scatterplot(X_test.ravel(), y_test)
sns.lineplot(X_test.ravel(), model.predict(poly.fit_transform(X_test))
)
```

Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1e018ac8>



In [61]:
```
lm = LinearRegression().fit(np.vander(x_train, N=2), (y_train - 15) /
14)
plot_partial_dependence(lm, np.vander(x_test, N=2), [0])
plt.xlabel('income06');
```



Based on above graph, we find that a negative relationship between income06 and the marginal effect of income06. As income06 increases, the marginal effect of income06 on egalit_scale decreases.

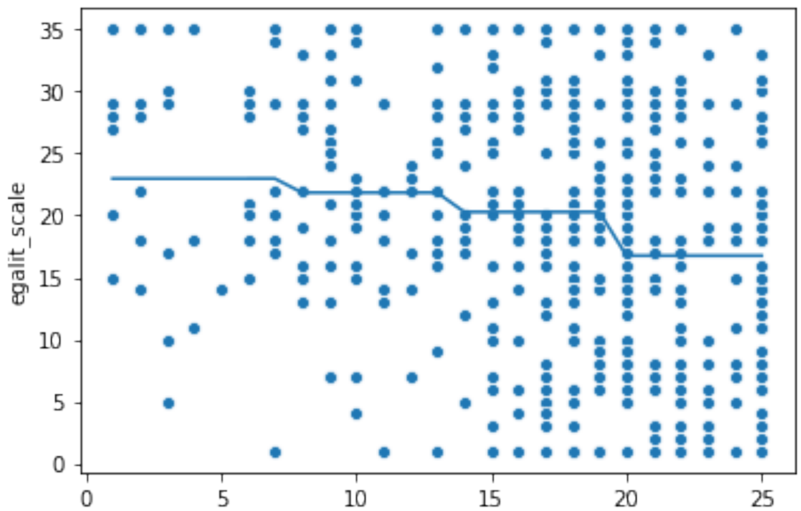
1.2 Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.

In [70]:
```
step_dict = {}
for i in range(1,11):
    x_cut, bins = pd.cut(X_train.ravel(), i, retbins = True, right = True)
    step_dummies = pd.get_dummies(x_cut)
    step_dummies = sm.add_constant(df_dummies)
    step_model = lm.fit(step_dummies, y_train)
    scores = cross_val_score(step_model, step_dummies, y_train,scoring="neg_mean_squared_error", cv=10)
    step_dict[i] = scores.mean()
```

The optimal number of cuts for step function is 4.

In [75]:
```
x_cut, bins = pd.cut(X_train.ravel(), 4, retbins = True, right = True)
step_dummies = pd.get_dummies(df_cut)
step_dummies = sm.add_constant(df_dummies)
step_model = lm.fit(df_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)
sns.lineplot(X_test.ravel(), model.predict(test_dummies))
```

Out[75]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1e073da0>



Based on above graph, we find that step function with 4 bins of income06 is the best fit. As the income increases, their degrees of egalitarian decreases.

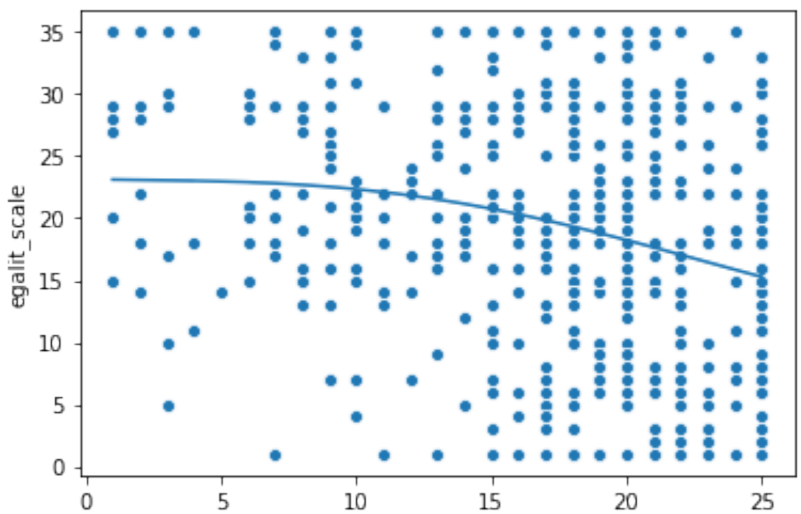
1.3 Fit a natural regression spline to predict egalit_scale as a function of income06. Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the results of the optimal model.

In [76]:
```
spline_dict = {}
for i in range(3, 11):
    transformed_x = dmatrix(f"cr(x, df={i}) - 1", {"x": x_train}, retu
rn_type='dataframe')
    model = lm.fit(transformed_x, y_train)
    scores = cross_val_score(model, transformed_x, y_train,scoring="ne
g_mean_squared_error", cv=10)
    spline_dict[i] = np.mean(scores)
print('the optimal degree of freedom:', max(spline_dict, key=spline_di
ct.get))
```

```
the optimal degree of freedom: 4
```

In [80]:
```
transformed_x = dmatrix("bs(train, df=4, degree=4)", {"train": X_train
},return_type='dataframe')
model = lm.fit(transformed_x, y_train)
test_spline = dmatrix("bs(test, df=4, degree=4)", {"test": X_test},ret
urn_type='dataframe')
sns.scatterplot(X_test.ravel(), y_test)
sns.lineplot(X_test.ravel(), model.predict(test_spline))
```

Out[80]: <matplotlib.axes._subplots.AxesSubplot at 0x1c1d3b7cc0>



Based on above graph, we find that the optimal degree of freedom is 4. As income06 increases, the value of egalit scale decreases.

1.4 (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):

In [81]:
```
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 [86]:
```
scaler = MinMaxScaler(feature_range=(0, 1))
def s_features(df):
    for column in df:
        if df[column].dtypes == object:
            df[column] = pd.get_dummies(df[column])
        elif df[column].dtypes == 'int64':
            transform_col = df[column].values.reshape(-1,1)
            scaler.fit(transform_col)
            df[column] = scaler.transform(transform_col)
    return df
X_train = s_features(X_train)
X_test = s_features(X_test)
transform_ytr = y_train.values.reshape(-1,1)
scaler.fit(transform_ytr)
y_train = scaler.transform(transform_ytr)
transform_yte = y_test.values.reshape(-1,1)
scaler.fit(transform_yte)
y_test = scaler.transform(transform_yte)
```

a. Linear regression

In [89]:
```
lm = LinearRegression().fit(X_train, y_train)
scores = cross_val_score(lm, X_train, y_train, scoring="neg_mean_squar
ed_error", cv=10)
lm_mse = np.mean(np.abs(scores))
print("Test MSE for Linear Regression: ", lm_mse)
```

```
Test MSE for Linear Regression:  0.05578476256153218
```

b. Elastic net regression

In [104]:
```
elastic = ElasticNetCV(cv=10, alphas=np.arange(0, 1.1, 0.1)).fit(X_tra
in, y_train)
elastic_mse = mean_squared_error(elastic.predict(X_test), y_test)
print('l1 ratio: ', elastic.l1_ratio_)
print('alpha: ', elastic.alpha_)
print('Test MSE of elastic net: ', elastic_mse)
```

```
l1 ratio:  0.5
alpha:  0.0
Test MSE of elastic net:  0.056237365591393106
```

c. Principal component regression

In [103]:
```
pcr_dict = {}
for i in np.arange(0.3, 1, 0.05):
    pca = PCA(i)
    x_reg = pca.fit_transform(X_train)
    reg = LinearRegression().fit(xreg, y_train)
    scores = cross_val_score(reg, xreg, y_train,scoring="neg_mean_squa
red_error", cv=10)
    pcr_mse = np.mean(np.abs(scores))
    pcr_dict[i] = pcr_mse

pca = PCA(min(pcr_dict, key=pcr_dict.get))
x_reg = pca.fit_transform(X_train)
reg = LinearRegression().fit(xreg, y_train)
scores = cross_val_score(reg, xreg, y_train, scoring="neg_mean_squared
_error", cv=10)
pcr_mse = np.mean(np.abs(scores))
print("Test MSE of principal component regression: ", pcr_mse)
```

```
the test MSE of principal component regression:  0.0556469139072822
```

d. Partial least squares regression

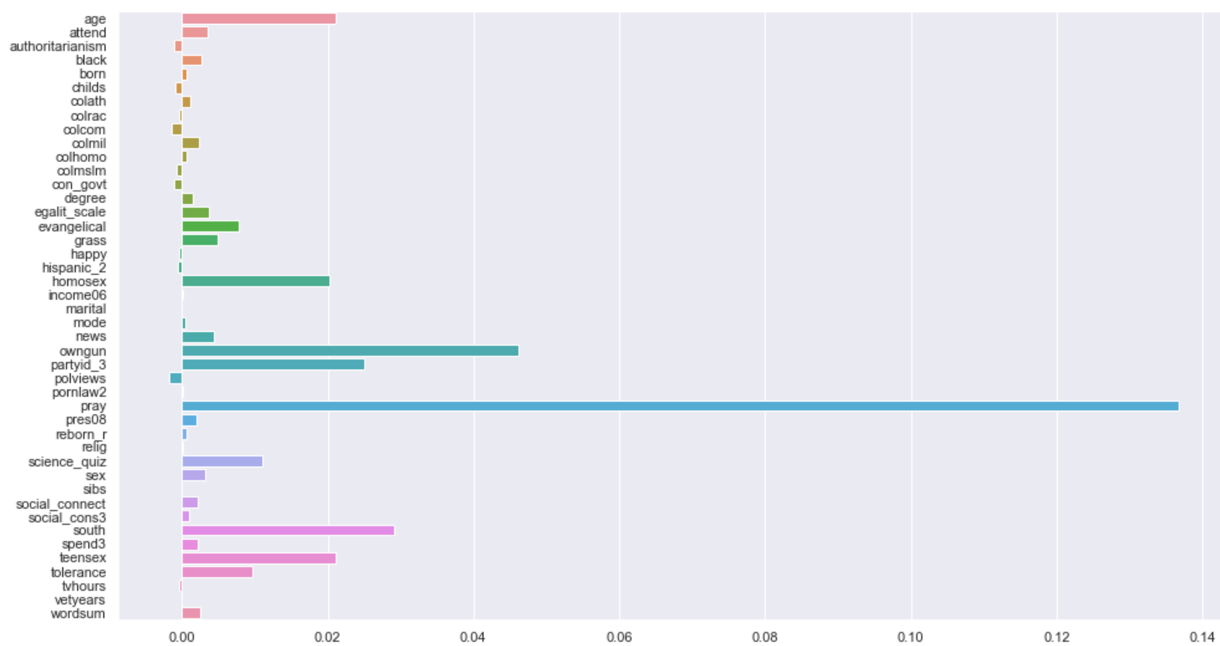
In [106]:
```
pls_dict = {}
for i in np.arange(1, 45):
    pls = PLSRegression(i).fit(X_train, y_train)
    scores = cross_val_score(pls, X_train, y_train,scoring="neg_mean_s
quared_error", cv=10)
    pls_mse = np.mean(np.abs(scores))
    pls_dict[i] = pls_mse

pls = PLSRegression(min(pls_dict, key=pls_dict.get)).fit(X_train, y_tr
ain)
scores = cross_val_score(pls, X_train, y_train,scoring="neg_mean_squar
ed_error", cv=10)
pls_mse = np.mean(np.abs(scores))
print("Test MSE of partial least squares: ", pls_mse)
```

```
Test MSE of partial least squares:  0.05572252235458728
```

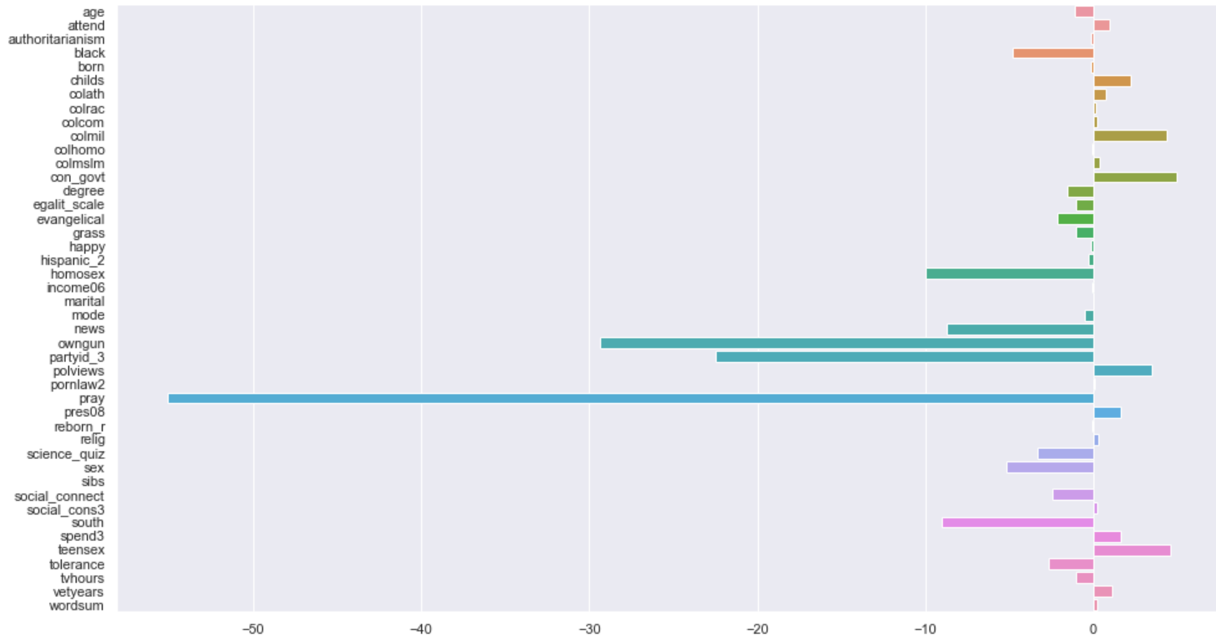
1.5 For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other (e.g., talk about feature importance, conditional effects, how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

In [120]:
```
imput = SimpleImputer(missing_values = np.nan, strategy = 'mean',verbose=0)
imput = imput.fit(X_test)
imput_test = imput.transform(X_test)

def plot_imp(model):
    imput_vals, _ = feature_importance_permutation(
        predict_method=model.predict,
        X=X_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(imput_vals[i])
    plt = sns.barplot(x=imp, y=col)
    return plt
```

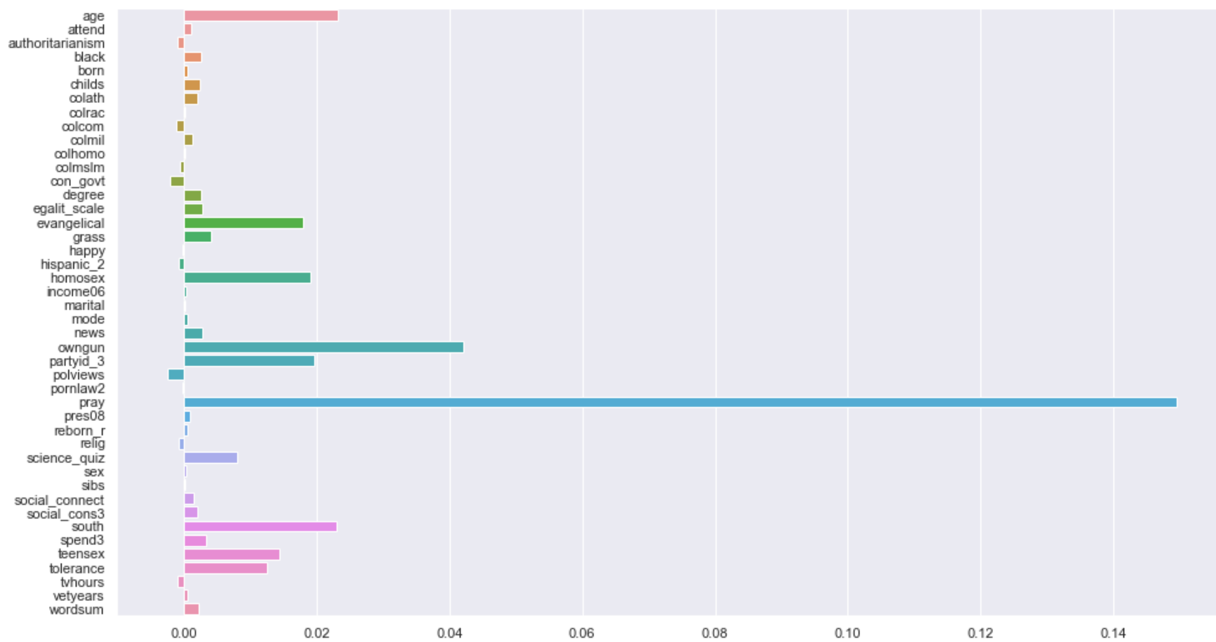
In [121]:
```
lm_plot = plot_imp(lm)
```



In [129]:
```
elastic_plot = plot_imp(elastic)
```

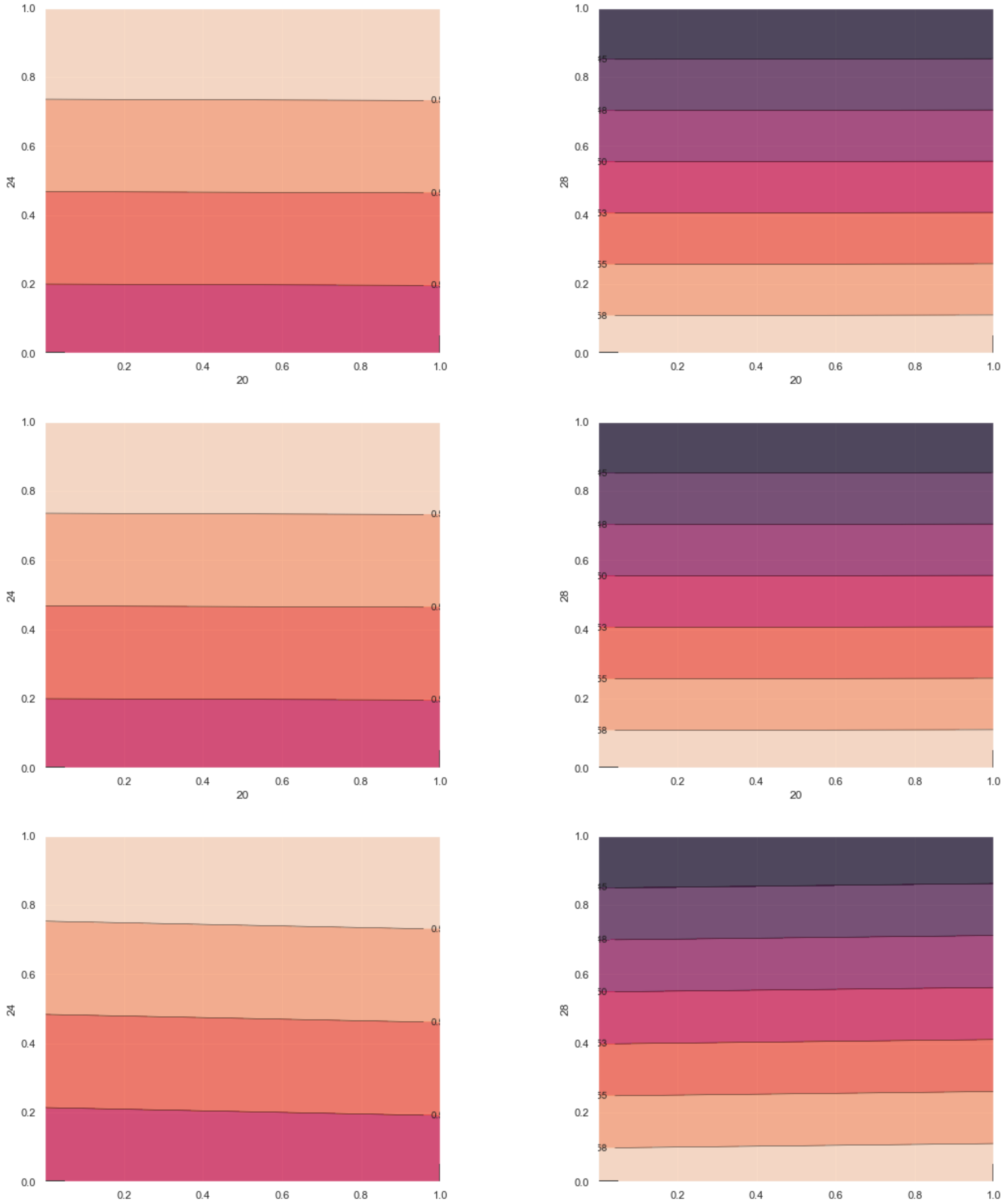


In [124]:
```
pls_plot = plot_imp(pls)
```



Based on above graphs, we can find feature importance plots of different methods. They are ranked differently across different methods because the parameters in each model are different.However, variables including pray, owngun, south are important features in all models.

In [133]:
```
#24=owngun, 28=pray,20=income06
plot_partial_dependence(lm, X_train, [(20,24), (20,28)])
plot_partial_dependence(elastic, X_train, [(20,24), (20,28)])
plot_partial_dependence(pls, X_train, [(20,24), (20,28)])
```



24=owngun, 28=pray,20=income06 Based on above graphs, we find that owngun,pray play important roles in explaining the model. So I plot the interaction between income06 and these two variables for different models. There is interaction between income06 and owngun, but there isn't interaction between income06 and pray.