30100HW4

February 16, 2020

[388]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
import sklearn
from sklearn.base import BaseEstimator
import seaborn as sns
from sklearn.inspection import plot_partial_dependence, partial_dependence
from statsmodels.tools.tools import add_constant
import statsmodels.api as sm
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
```

1 1

[389]:
```
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)
```

[390]:
```
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
x_train = gss_train['income06']
x_test = gss_test['income06']
```

[392]:
```
class PolynomialRegression(BaseEstimator):
    def __init__(self, deg=None):
        self.deg = deg

    def fit(self, X, y):
        self.model = LinearRegression(fit_intercept=False)
        self.model.fit(np.vander(X, N=self.deg + 1), y)
```

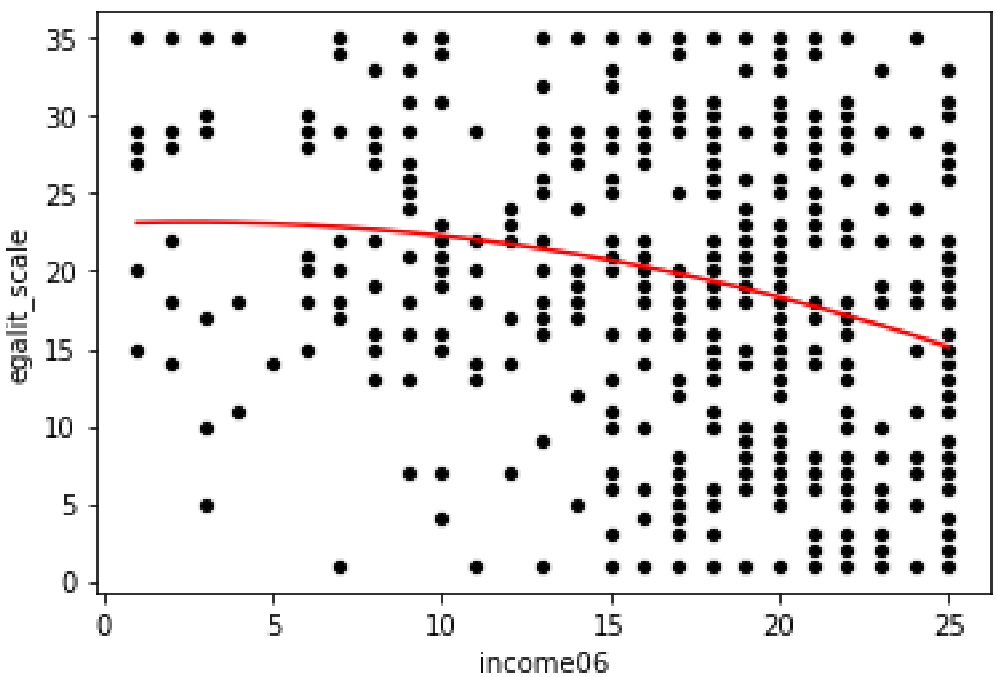
```python
    def predict(self, x):
        return self.model.predict(np.vander(x, N=self.deg + 1))

    @property
    def coef_(self):
        return self.model.coef_
```

[393]:
```python
estimator = PolynomialRegression()
degrees = np.arange(1, 6)
cv_model = GridSearchCV(estimator,
                        param_grid={'deg': degrees},
                        scoring='neg_mean_squared_error',
                       cv=10)
cv_model.fit(x_train, y_train);
```

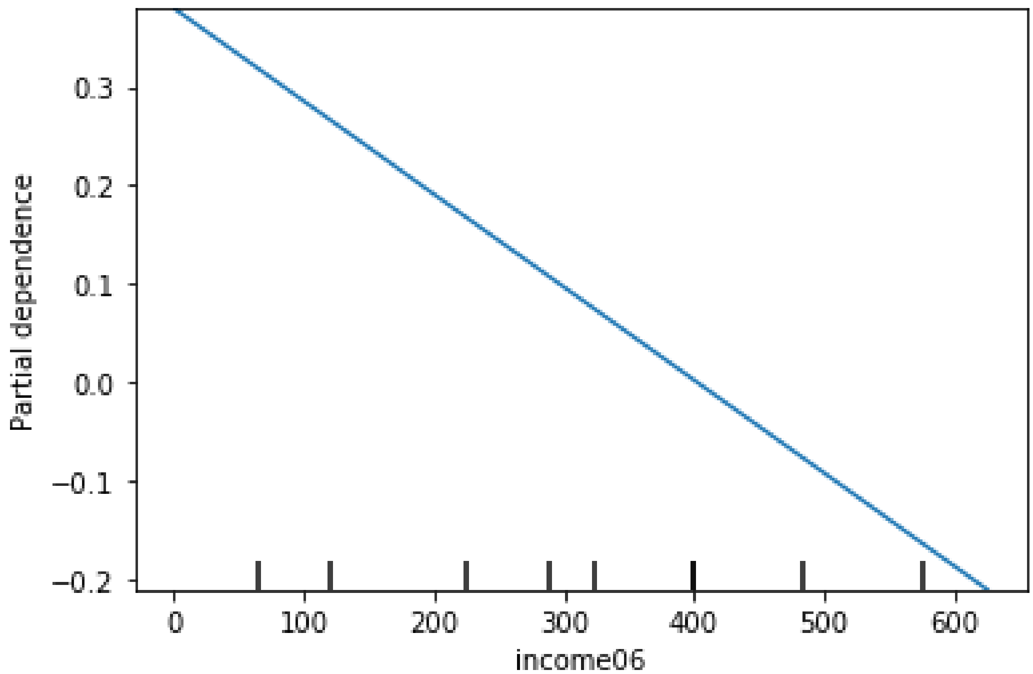
[394]:
```python
cv_model.best_params_, cv_model.best_estimator_.coef_
```

[394]:
```
({'deg': 2}, array([-1.60224341e-02,  8.35584082e-02,  2.30487704e+01]))
```

[395]:
```python
sns.scatterplot(x_test, y_test, color='black')
sns.lineplot(x_test, cv_model.predict(x_test), color='red');
```



[401]:
```python
lm = LinearRegression().fit(np.vander(x_train, N=3), (y_train - 18) / 17)
plot_partial_dependence(lm, np.vander(x_test, N=3), [0])
plt.xlabel('income06');
```



Through 10-fold cross validation, we find that the best polynomial model is a quadratic model as seen in the plot, and income06's partial dependence gradually decreases with a negative slope, which suggests that the marginal effect of income06 is initially slightly positive, and as the value increases, its marginal effect gradually decreases to a negative value.

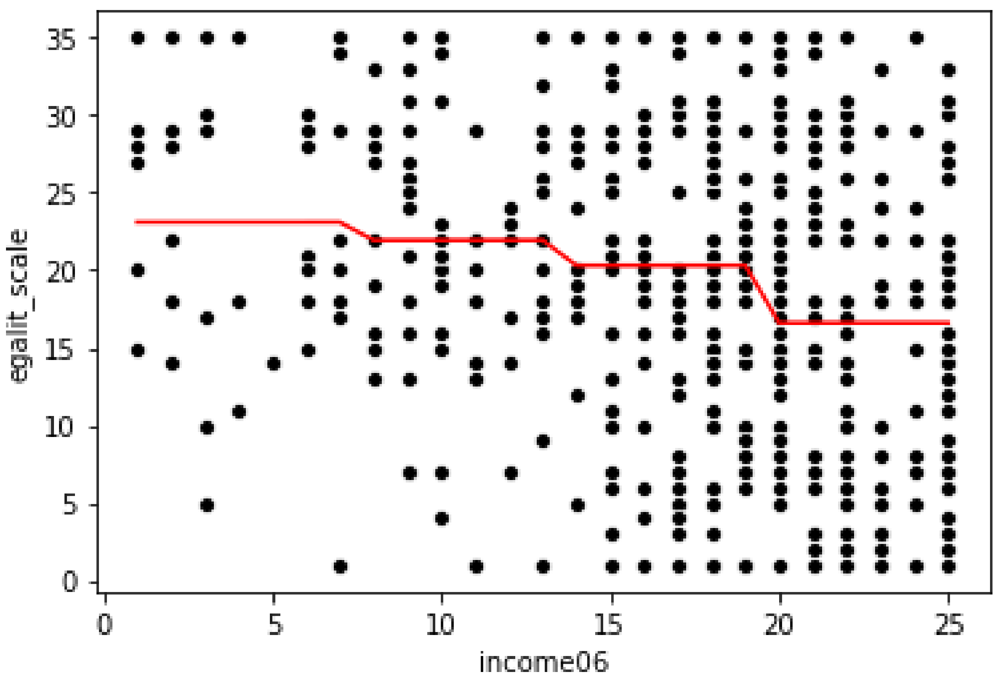
2 2

[282]:
```python
select_dict = {}
for i in range(1,11):
    x_train_cut, bins = pd.cut(x_train, bins=i, retbins=True)
    steps_dummies = pd.get_dummies(x_train_cut)
    step_model = LinearRegression().fit(steps_dummies, y_train)
    scores = cross_val_score(step_model, steps_dummies, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
    select_dict[i] = scores.mean()
print('the optimal number of cuts: ', max(select_dict, key=select_dict.get))
```

```
the optimal number of cuts:  4
```

[283]:
```
x_cut, bins = pd.cut(x_train, bins=4, retbins=True)
steps_dummies = pd.get_dummies(x_cut)
step_model = LinearRegression().fit(steps_dummies, y_train)
bin_mapping = np.digitize(x_test, bins, right=True)
test_steps_dummies = pd.get_dummies(bin_mapping)
sns.scatterplot(x_test, y_test, color='black')
sns.lineplot(x_test, step_model.predict(test_steps_dummies), color='red');
steps_dummies.columns
```

[283]:
```
CategoricalIndex([(0.976, 7.0], (7.0, 13.0], (13.0, 19.0], (19.0, 25.0]],
categories=[(0.976, 7.0], (7.0, 13.0], (13.0, 19.0], (19.0, 25.0]],
ordered=True, name='income06', dtype='category')
```



Through 10-fold cross-validation, we find that the step function with 4 bins can best fit the data, which suggests that basically the sample can be divided into four parts by their income06, and as the income increases, their degrees of egalitarian drops correspondingly.

3 3

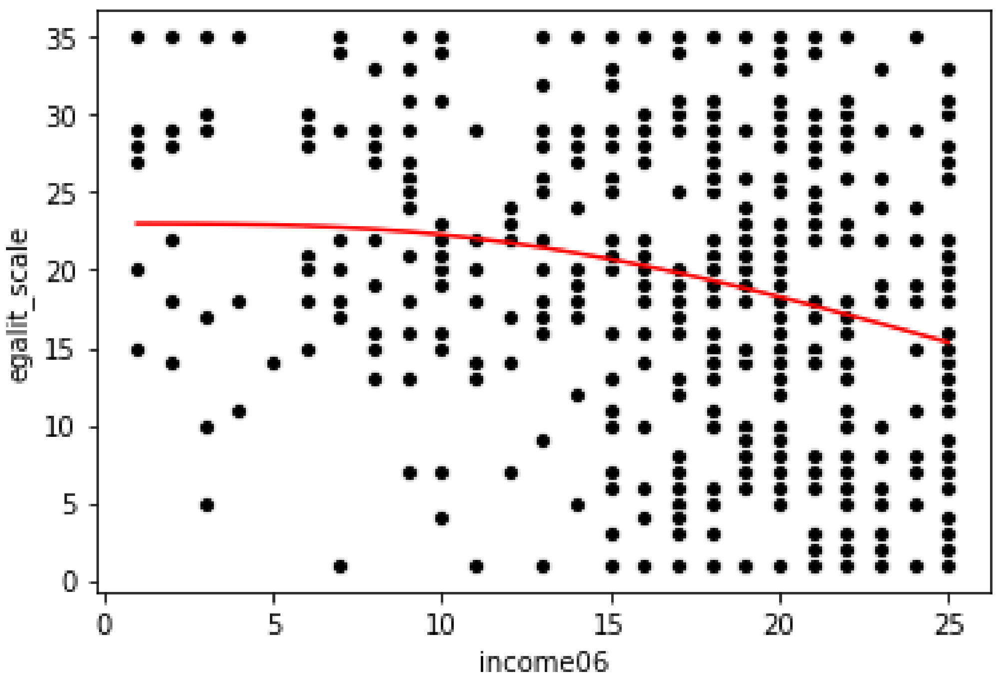
[256]:
```
from patsy import dmatrix
```

[289]:
```
spline_dict = {}
for i in range(3, 11):
```

```
    transformed_x = dmatrix(f"cr(x, df={i}) - 1", {"x": x_train},␣
↪return_type='dataframe')
    model = lm.fit(transformed_x, y_train)
    scores = cross_val_score(model, transformed_x, y_train,␣
↪scoring="neg_mean_squared_error", cv=10)
    spline_dict[i] = np.mean(scores)

print('the optimal degree of freedom:', max(spline_dict, key=spline_dict.get))
```

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

[292]:
```
transformed_x = dmatrix("cr(x, df=4) - 1", {"x": x_train},␣
↪return_type='dataframe')
ns_model = lm.fit(transformed_x, y_train)
transformed_test = dmatrix("cr(x, df=4) - 1", {"x": x_test},␣
↪return_type='dataframe')
sns.scatterplot(x_test, y_test, color='black')
sns.lineplot(x_test, ns_model.predict(transformed_test), color='red');
```



Through 10-fold cross validation, we can find that the optimal degree of freedom is 4 whereby we can draw a much smoother curve than the step function, but a bit similar to the polynomial curve. Considering that the scale of the x-axis is not large enough to show the oscillation of the polynomial curve, the similarity of the results is reasonable.

4 4

[293]:
```
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)
```

[302]:
```
scaler = MinMaxScaler(feature_range=(0, 1))

def scale_features(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
```

[303]:
```
x_train = scale_features(x_train)
x_test = scale_features(x_test)
y_train = scale_features(y_train)
y_test = scale_features(y_test)
```

4.1 a

[304]:
```
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:  0.055784762561532183
```

4.2 b

[313]:
```
from  sklearn.linear_model import ElasticNetCV
import warnings
warnings.filterwarnings("ignore")
alpha = np.arange(0, 1.1, step=0.1)
en = ElasticNetCV(cv=10, alphas=alpha).fit(x_train, y_train)
en_mse = mean_squared_error(en.predict(x_test), y_test)
```

```
print('l1 ratio: ', en.l1_ratio_)
print('alpha: ', en.alpha_)
print('the test MSE of elastic net: ', en_mse)
```

```
l1 ratio:  0.5
alpha:  0.0
the test MSE of elastic net:  0.0562373655913931
```

4.3 c

[330]:
```
pcr_dict = {}
for i in np.arange(0.3, 1, 0.05):
    pca = PCA(i)
    xreg = 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))
    pcr_dict[i] = pcr_mse

min(pcr_dict, key=pcr_dict.get)
```

[330]:
```
0.7
```

[327]:
```
pca = PCA(0.7)
xreg = 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("the test MSE of principal component regression: ", pcr_mse)
```

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

4.4 d

[332]:
```
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)
```

[332]:
```
6
```

[337]:
```
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: ", pls_mse)
```

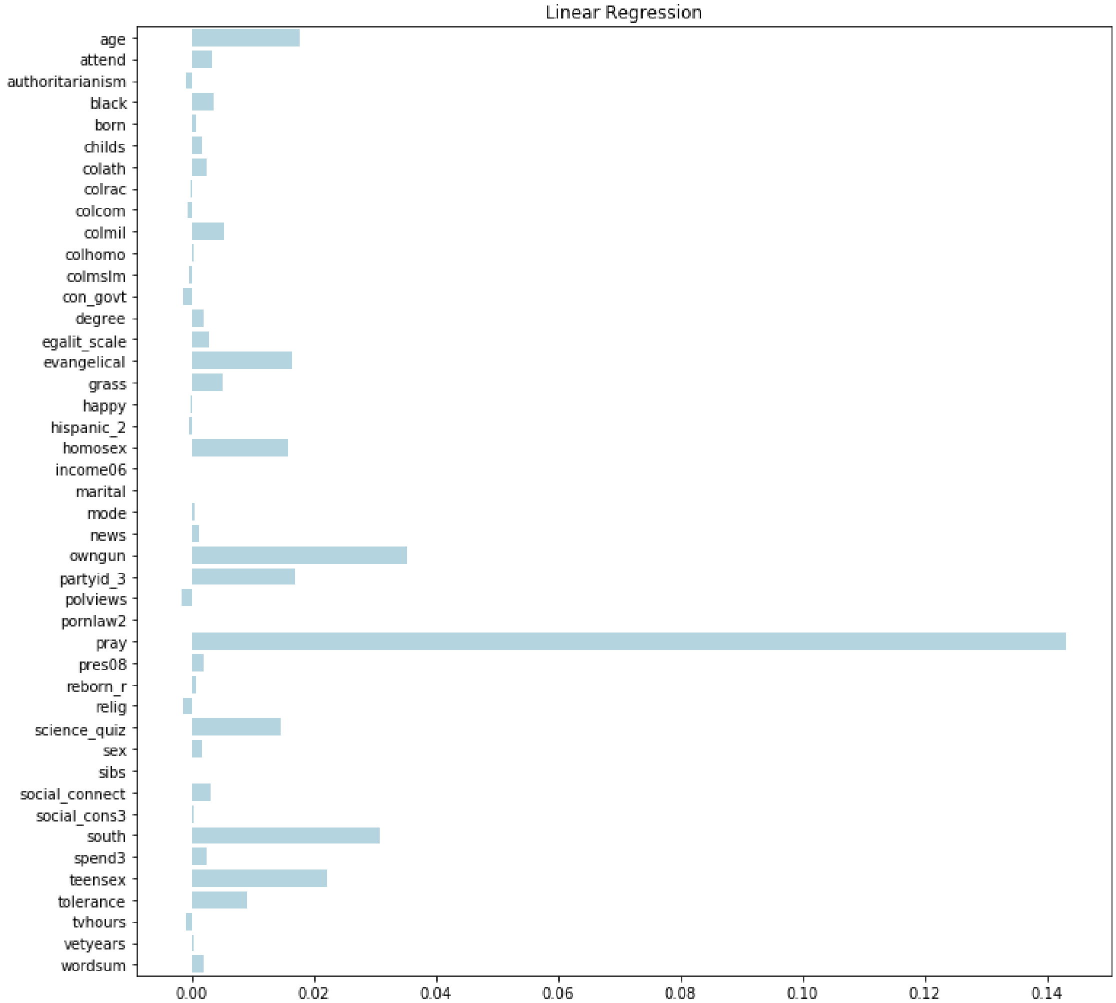
```
the test MSE of partial least squares:  0.05572252235458729
```

5 5

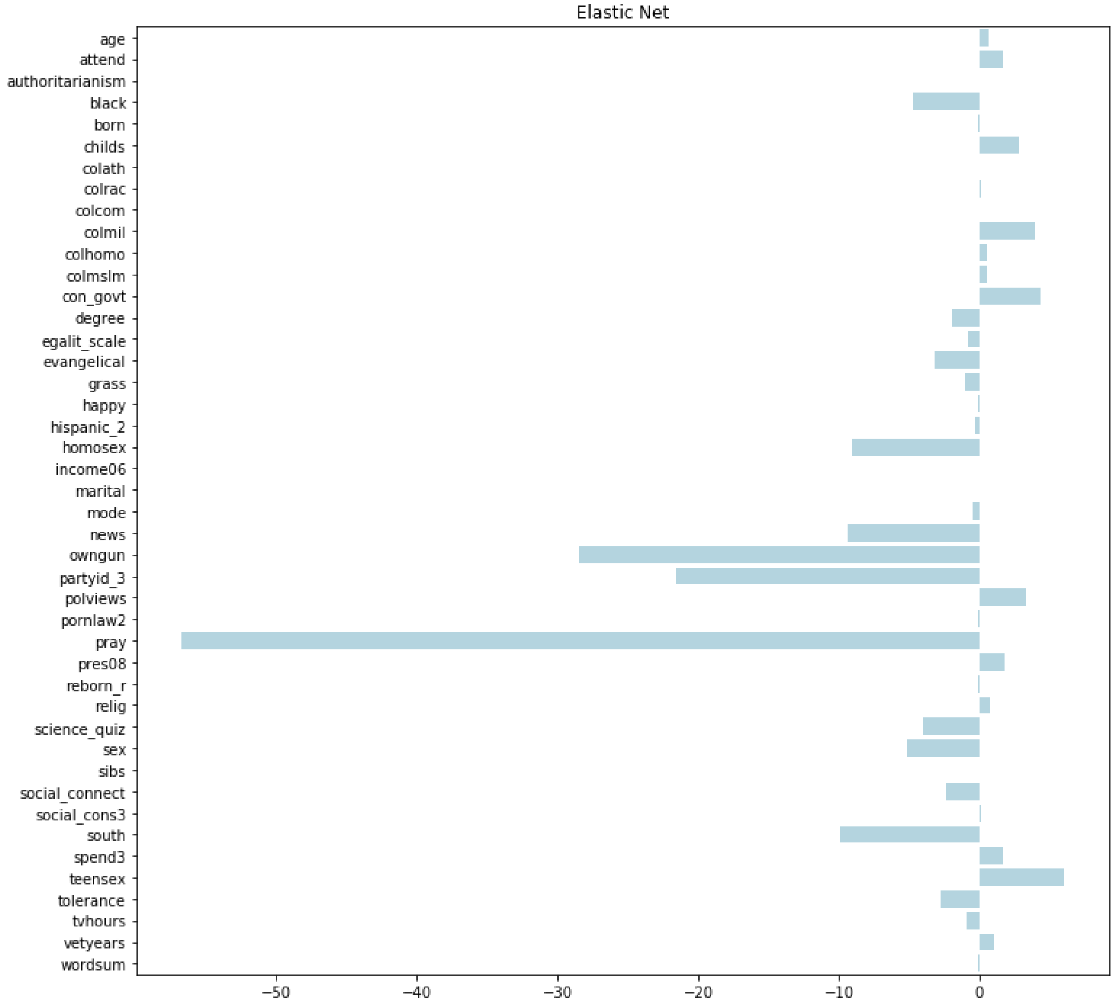
[338]:
```
from mlxtend.evaluate import feature_importance_permutation
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean', verbose=0)
imputer = imputer.fit(x_test)
impute_test = imputer.transform(x_test)
```

[348]:
```
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='lightblue')

    return ax
```

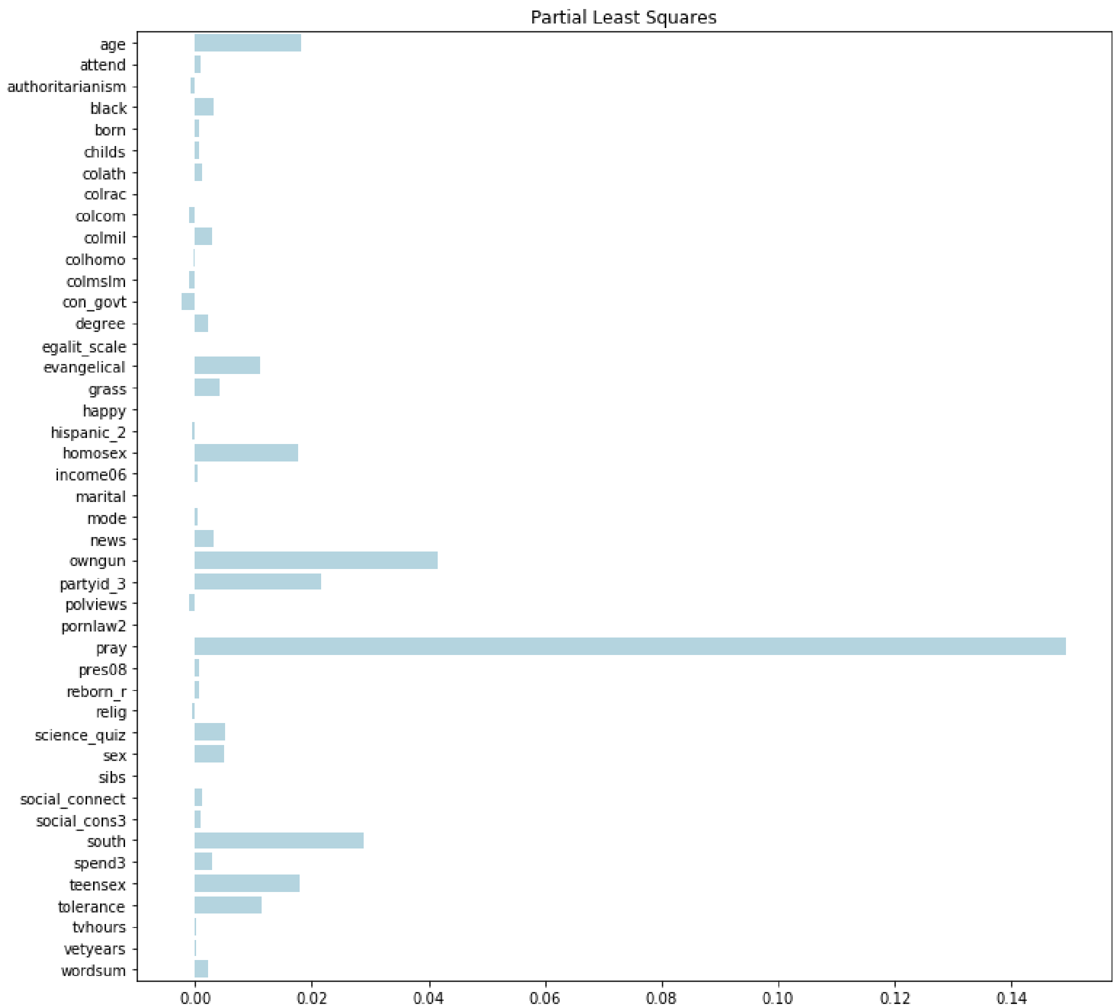
[350]:
```
plt.figure(figsize=(12, 12))
lm_plot = plot_imp(lm)
plt.title('Linear Regression');
```



[351]:
```
plt.figure(figsize=(12, 12))
elasticnet_plot = plot_imp(en)
plt.title('Elastic Net');
```

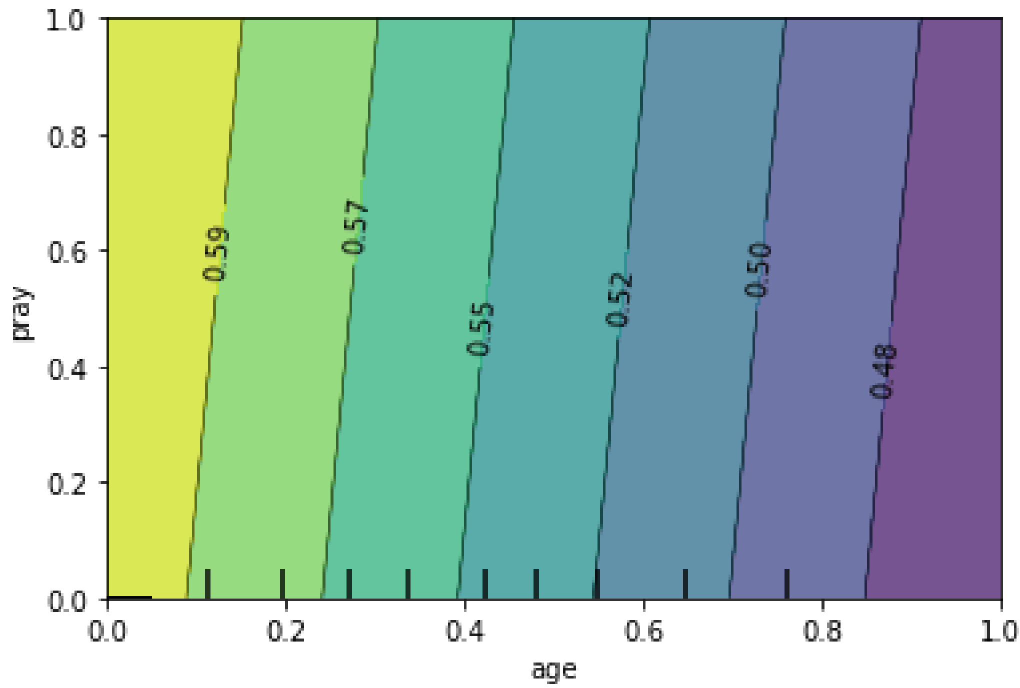


[352]:
```
plt.figure(figsize=(12, 12))
elasticnet_plot = plot_imp(pls)
plt.title('Partial Least Squares');
```



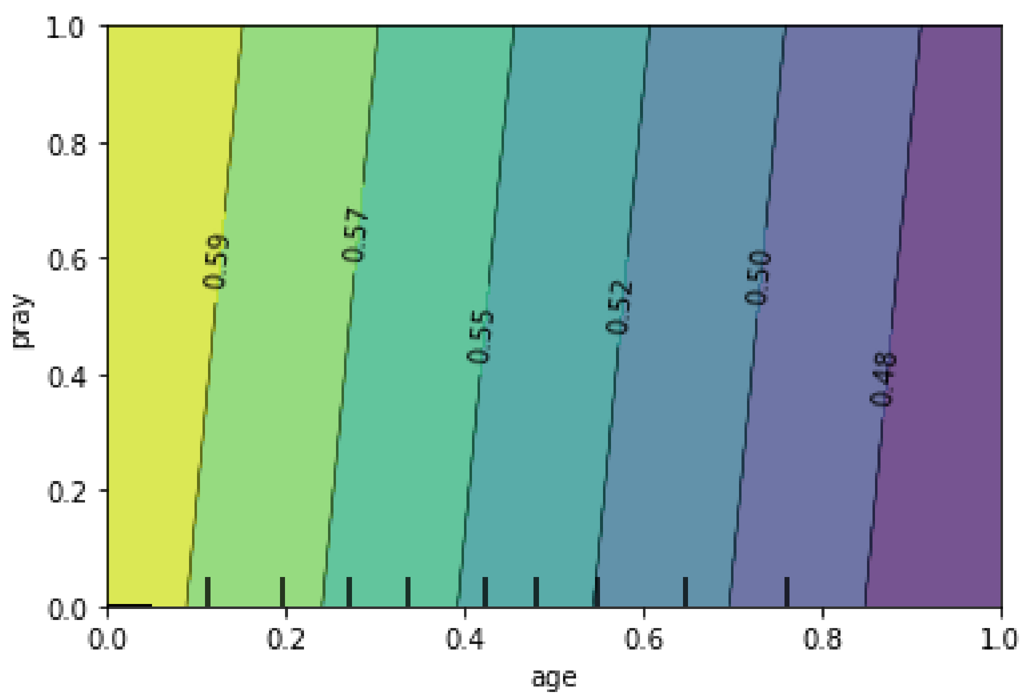
Above are the feature importance plots of different methods. They are ranked very differently or even contrarily because each regression method selects the parameters by taking different aspects of the models into consideration, like penalizing different aspects. But there are some common patterns existing in all these plots. The variables pray, age, owngun, teensex all play a significant role in explaining the models. We take pray and age for example to investigate their feature interaction.

[374]:
```
plot_partial_dependence(lm, x_test, [(0, 27)]);
```



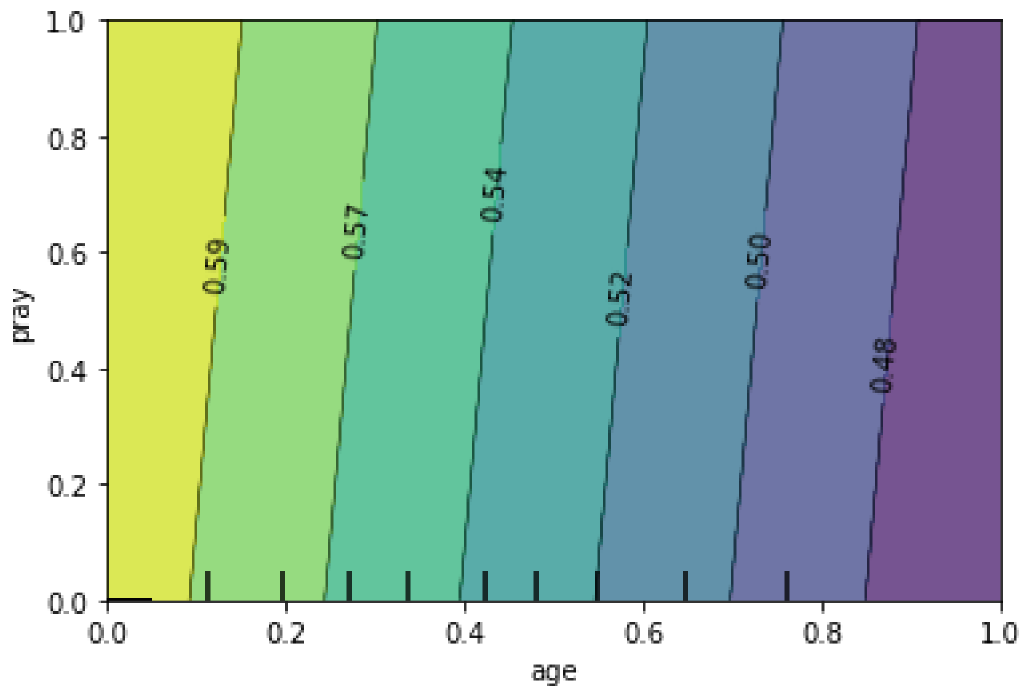
[366]:
```
plot_partial_dependence(en, x_test, [(0, 27)])
```

[366]: <sklearn.inspection._partial_dependence.PartialDependenceDisplay at 0x1e461e9e460>



[367]:
```
plot_partial_dependence(pls, x_test, [(0, 27)])
```

[367]: <sklearn.inspection._partial_dependence.PartialDependenceDisplay at 0x1e461ed7700>



Surprisingly, contrary to the previous results, partial dependence remains relatively stable across regression methods for age and pray. According to the plots, there seem to be no nonlinear interactions between pray and age, which indicates that age and the frequency of pray have a linear relationship.