In [261]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import statsmodels.api as sm
```

In [357]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [262]:
```python
train = pd.read_csv('gss_train.csv')
test = pd.read_csv('gss_test.csv')
train.head()
```

Out[262]:

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

5 rows × 45 columns

In [263]:
```python
from sklearn.model_selection import KFold
kf_10 = KFold(n_splits=10, shuffle=False, random_state=1)
```

Egalitarianism and income

1 Polynominal Regression

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.

- The average marginal effect of x on y is just dy/dx. You can use numpy's np.gradient() function to calculate this on the model's predicted y over some range of x values.

Be sure to provide substantive interpretation of the results.

In [304]:

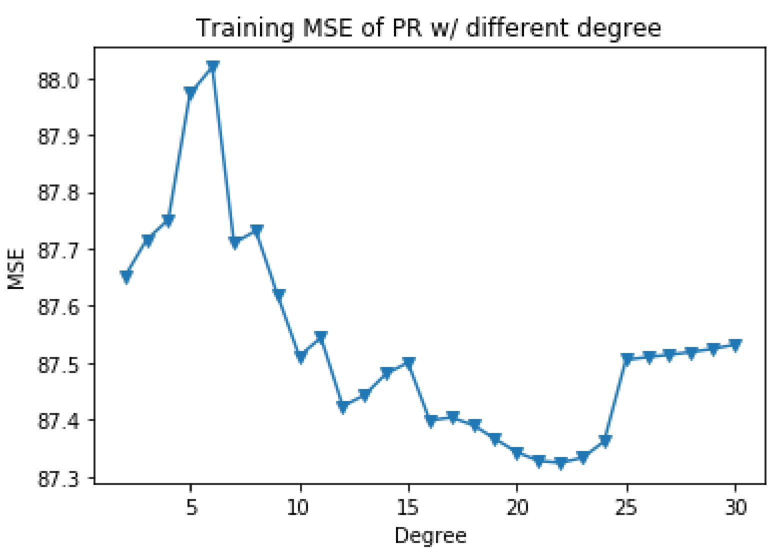
```
from sklearn.linear_model import LinearRegression
regr = LinearRegression()

from sklearn.linear_model import LinearRegression

from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
```

In [270]:

```
train_x = train.income06
train_y = train.egalit_scale
test_x = test.income06
test_y = test.egalit_scale
```

In [312]:
```
# 10-fold CV
pr_mse = []

# Calculate MSE using CV for the 19 principle components,
# adding one component at the time.
for i in range(2, 31):
    x_features = PolynomialFeatures(degree=i, include_bias=False)
    train_x_t = x_features.fit_transform(train_x.values.reshape(-1,1))
#     test_x_t = x_features.fit_transform(test_x.values.reshape(-1,1))
    score = cross_val_score(regr, train_x_t, train_y.ravel(),
                            cv=kf_10, scoring='neg_mean_squared_error').mean()
    pr_mse.append(-score)

plt.plot(range(2, 31), pr_mse, '-v')
plt.xlabel('Degree')
plt.ylabel('MSE')
plt.title('Training MSE of PR w/ different degree')

d = pr_mse.index(min(pr_mse))
print(f'The best degree is {d+1}')
```

```
The best degree is 21
```



GridSearchCV

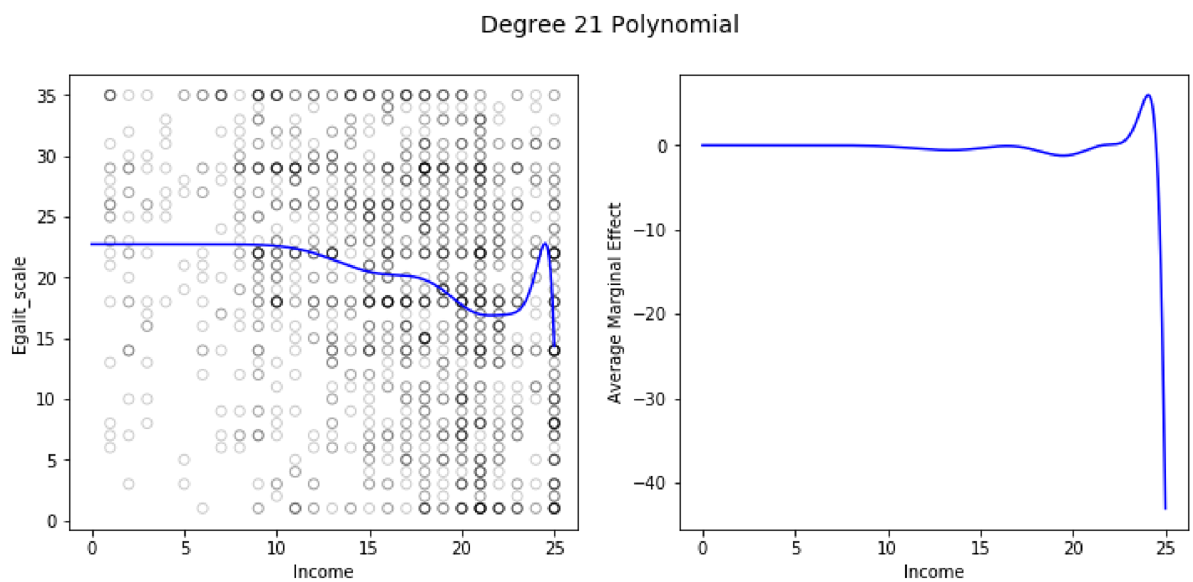
another way to find the optimal degree

In [404]:
```
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV

# Build a pipeline:
def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))

# Use GridSearchCV to find the optimal degree through 10-fold CV
# Define the GridSearchCV parameters:
param_grid = {'polynomialfeatures__degree': np.arange(30),
              'linearregression__fit_intercept': [True, False],
              'linearregression__normalize': [True, False]}
grid = GridSearchCV(PolynomialRegression(), param_grid, cv=10)
grid.fit(train_x.values.reshape(-1,1), train_y)

# get the best parameters
model = grid.best_estimator_
model
```

Out[404]:
```
Pipeline(memory=None,
         steps=[('polynomialfeatures',
                 PolynomialFeatures(degree=22, include_bias=True,
                                    interaction_only=False, order
='C')),
                ('linearregression',
                 LinearRegression(copy_X=True, fit_intercept=True, n_jo
bs=None,
                                  normalize=False))],
         verbose=False)
```

In [342]:

```python
# sns.regplot(train.income06, train.egalit_scale, order = 21,
# truncate=True, scatter=False);

x_features = PolynomialFeatures(degree=21, include_bias=False)
train_x_t = x_features.fit_transform(train_x.values.reshape(-1,1))
mod = regr.fit(train_x_t, train_y)

x_grid = np.linspace(0,25,1000)
x_t = x_features.fit_transform(x_grid.reshape(-1,1))
pred = mod.predict(x_t)

dx = x_grid[1] - x_grid[0]
ame = np.gradient(pred, dx)

fig, (ax1, ax2) = plt.subplots(1,2, figsize=(12,5))
fig.suptitle('Degree 21 Polynomial', fontsize=14)

ax1.scatter(train.income06, train.egalit_scale,
            facecolor='None', edgecolor='k', alpha=0.2)
ax1.plot(x_grid, pred, 'b')
ax1.set_xlabel('Income')
ax1.set_ylabel('Egalit_scale');

ax2.plot(x_grid, ame, 'b')
ax2.set_xlabel('Income')
ax2.set_ylabel('Average Marginal Effect');
```



Reference:

https://stackoverflow.com/questions/47442102/how-to-find-the-best-degree-of-polynomials

https://nbviewer.jupyter.org/github/JWarmenhoven/ISL-python/blob/master/Notebooks/Chapter%207.ipynb#7.8.1-Polynomial-Regression-and-Step-Functions

2 Step Function

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 [390]:
```
df_cut, bins = pd.cut(train_x, 4, retbins=True, right=True)
df_cut.value_counts(sort=False)
```

Out[390]:
```
(0.976, 7.0]     108
(7.0, 13.0]      303
(13.0, 19.0]     504
(19.0, 25.0]     566
Name: income06, dtype: int64
```

In [391]:
```
df_steps = pd.concat([train_x, df_cut, train_y],
                     keys=['income','income_cuts','egalit_scale'], axis=
1)
df_steps.head(5)
```

Out[391]:

	income	income_cuts	egalit_scale
0	25	(19.0, 25.0]	22
1	23	(19.0, 25.0]	14
2	19	(13.0, 19.0]	20
3	16	(13.0, 19.0]	34
4	5	(0.976, 7.0]	35

In [392]:
```
# Create dummy variables for the income groups
df_steps_dummies = pd.get_dummies(df_steps['income_cuts'])
# Statsmodels requires explicit adding of a constant (intercept)
df_steps_dummies = sm.add_constant(df_steps_dummies)
df_steps_dummies.head(5)
```

Out[392]:

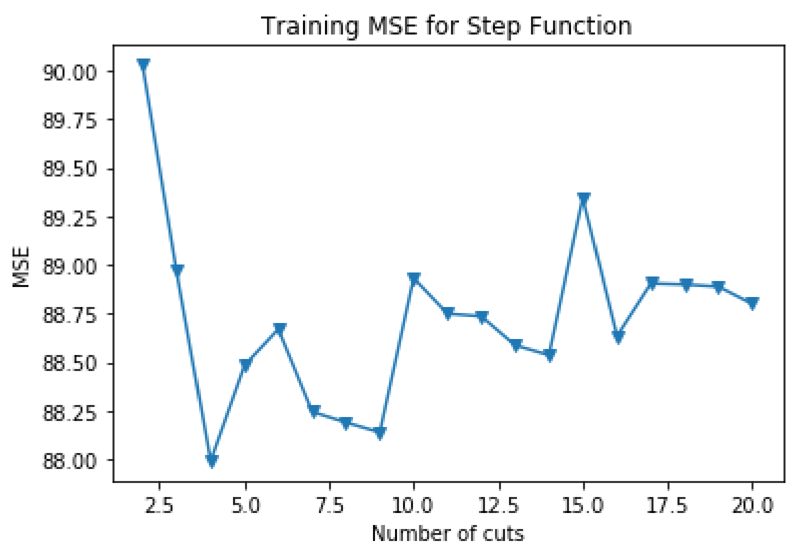
	const	(0.976, 7.0]	(7.0, 13.0]	(13.0, 19.0]	(19.0, 25.0]
0	1.0	0	0	0	1
1	1.0	0	0	0	1
2	1.0	0	0	1	0
3	1.0	0	0	1	0
4	1.0	1	0	0	0

In [ ]:
```
sf = regr.fit(df_steps_dummies.drop(df_steps_dummies.columns[1], axis=1
),
              y_train.ravel())
```

In [358]:

```python
sf_mse = []

for i in range(2,21):
    df_cut, bins = pd.cut(train_x, i, retbins=True, right=True)
    df_steps = pd.concat([train_x, df_cut, train_y],
                         keys=['income','income_cuts','egalit_scale'],
                         axis=1)

    # Create dummy variables for the income groups
    df_steps_dummies = pd.get_dummies(df_steps['income_cuts'])
    # Statsmodels requires explicit adding of a constant (intercept)
    df_steps_dummies = sm.add_constant(df_steps_dummies)

    score = cross_val_score(regr,
            df_steps_dummies.drop(df_steps_dummies.columns[1], axis=1),
            y_train.ravel(),
            cv=kf_10, scoring='neg_mean_squared_error').mean()
    sf_mse.append(-score)

plt.plot(range(2,21), sf_mse, '-v')
plt.xlabel('Number of cuts')
plt.ylabel('MSE')
plt.title('Training MSE for Step Function')
#plt.xlim(xmin=-1);
d = sf_mse.index(min(sf_mse))
print(f'The best number of cuts is {d+2}')
```

```
The best number of cuts is 4
```



In [374]:

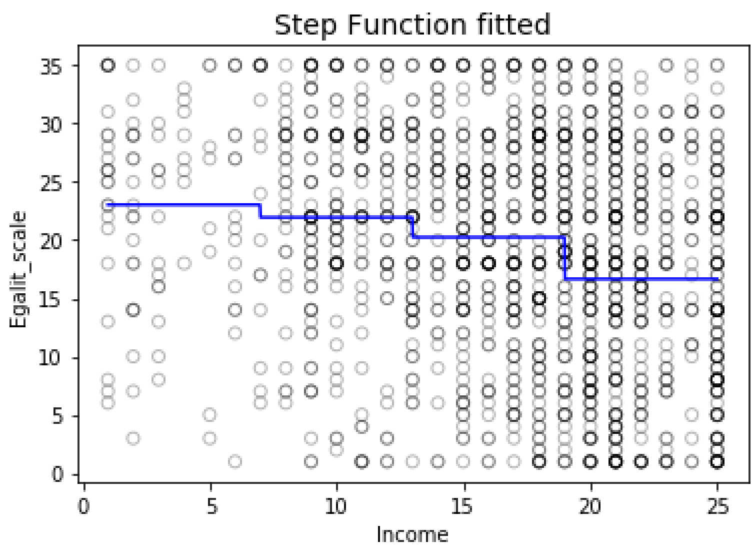
```python
bins
```

Out[374]:

```
array([ 0.976,  7.   , 13.   , 19.   , 25.   ])
```

In [399]:
```
x_grid = np.linspace(1, 24.999, 1000)
bin_mapping = np.digitize(x_grid.ravel(), bins)
df_steps_dummies = pd.get_dummies(bin_mapping)
df_steps_dummies = sm.add_constant(df_steps_dummies)
pred = sf.predict(df_steps_dummies.drop(df_steps_dummies.columns[1],
                                        axis=1))

plt.scatter(train_x, train_y, facecolor='None', edgecolor='k', alpha=0.3
)
plt.plot(x_grid, sf_pred, c='b')

plt.title('Step Function fitted', fontsize=14)
plt.xlabel('Income')
plt.ylabel('Egalit_scale')
```

Out[399]:
```
Text(0, 0.5, 'Egalit_scale')
```



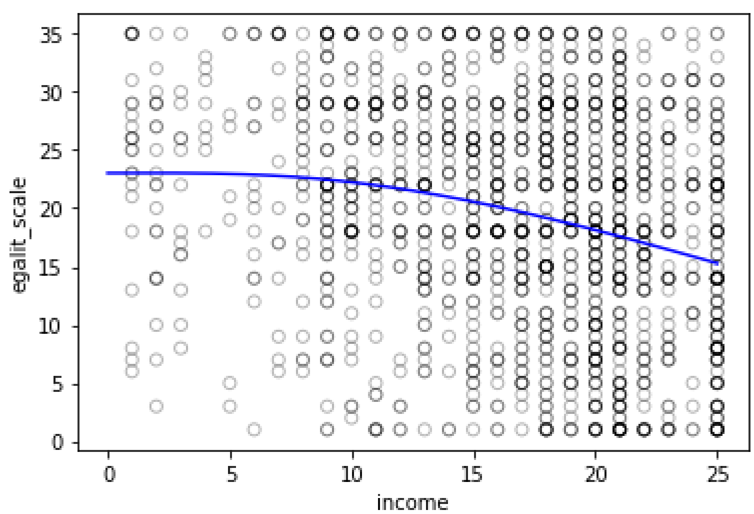
3 Natural regression spline

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 [423]:
```python
from patsy import dmatrix

x_grid = np.linspace(0,25,1000)
x_t = dmatrix("cr(train_x, df=4)", {"train_x": train_x},
              return_type='dataframe')
nrs = sm.GLM(train_y, x_t).fit()
nrs_pred = nrs.predict(dmatrix("cr(x_grid, df=4)",
                               {"x_grid": x_grid},
                               return_type='dataframe'))
nrs.params
```

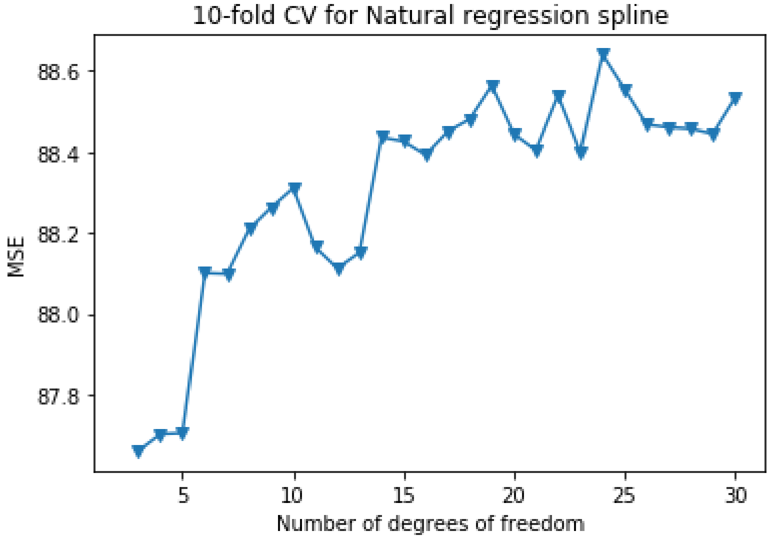
Out[423]:
```
Intercept                16.134294
cr(train_x, df=4)[0]      6.878604
cr(train_x, df=4)[1]      6.444906
cr(train_x, df=4)[2]      3.679405
cr(train_x, df=4)[3]     -0.868622
dtype: float64
```

In [424]:
```python
plt.scatter(train_x, train_y, facecolor='None', edgecolor='k', alpha=0.3
)

plt.plot(x_grid, nrs_pred, color='b', label='Natural spline df=4')
# plt.xlim(15,85)
# plt.ylim(0,350)
plt.xlabel('income')
plt.ylabel('egalit_scale');
```

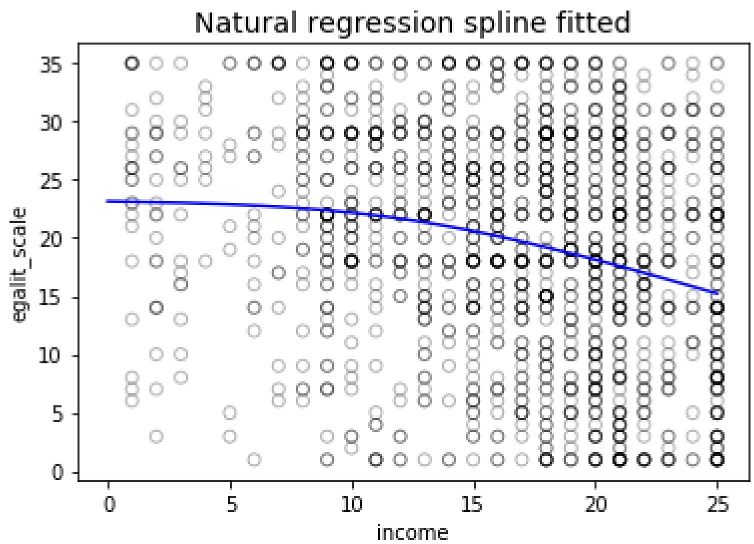


In [431]:
```
nrs_mse = []

for d in np.arange(3, 31):
    x_t = dmatrix(f"cr(train_x, df={d})", data={"train_x": train_x},
                  return_type='dataframe')

    score = -1*cross_val_score(regr, x_t, train_y.ravel(),
                      cv=kf_10, scoring='neg_mean_squared_error').mean()
    nrs_mse.append(score)

plt.plot(range(3, 31), nrs_mse, '-v')
plt.xlabel('Number of degrees of freedom')
plt.ylabel('MSE')
plt.title('10-fold CV for Natural regression spline')
plt.xlim(xmin=1);
index = nrs_mse.index(min(nrs_mse))
print(f'The best degrees of freedom is {index+3}')
```

```
The best degrees of freedom is 3
```



In [434]:
```
x_grid = np.linspace(0,25,1000)
x_t = dmatrix("cr(train_x, df=3)", {"train_x": train_x},
              return_type='dataframe')
nrs = sm.GLM(train_y, x_t).fit()
nrs_pred = nrs.predict(dmatrix("cr(x_grid, df=3)",
            {"x_grid": x_grid}, return_type='dataframe'))

plt.scatter(train_x, train_y, facecolor='None',
            edgecolor='k', alpha=0.3)
plt.plot(x_grid, nrs_pred, color='b', label='Natural spline df=4')
plt.xlabel('income')
plt.ylabel('egalit_scale')
plt.title('Natural regression spline fitted', fontsize=14);
```



Egalitarianism and everything

In [435]:
```
train = train.select_dtypes(include='int64')
train.head()
```

Out[435]:

	age	authoritarianism	childs	con_govt	egalit_scale	income06	science_quiz	sibs	social_con
0	21	4	0	4	22	25	7	2	
1	42	4	2	2	14	23	10	1	
2	70	1	3	4	20	19	4	0	
3	35	2	2	2	34	16	7	2	
4	24	6	3	3	35	5	5	2	

In [436]:
```
test = test.select_dtypes(include='int64')
```

In [437]:
```
X_train, y_train = train.drop('egalit_scale', axis=1), \
                   train['egalit_scale']
X_test, y_test = test.drop('egalit_scale', axis=1), test['egalit_scale']
```

In [439]:
```
# data preprocessing: feature standardization
from sklearn import preprocessing

n_train = preprocessing.scale(X_train)
n_test = preprocessing.scale(X_test)

cv = {}
```

3a linear regression

In [440]:
```
m1 = LinearRegression().fit(n_train, y_train)

lr_cv_score = -1 * cross_val_score(m1, n_test, y_test,
                    cv=kf_10, scoring='neg_mean_squared_error').mean()
cv['Linear regression'] = lr_cv_score
```

3b Elastic net regression

In [441]:
```
from sklearn.linear_model import ElasticNetCV
cv_values = np.arange(0.1, 1.1, 0.1)

m2 = ElasticNetCV(l1_ratio=cv_values, cv=10, random_state=1917)
m2.fit(n_train, y_train)

en_cv_score = -1 * cross_val_score(m2, n_test, y_test,
                    cv=kf_10, scoring='neg_mean_squared_error').mean()
cv['Elastic net regression'] = en_cv_score
```

3c Principal component regression

In [186]:
```
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
```

In [442]:
```
pca = PCA()
X_reduced = pca.fit_transform(n_train)

print(pca.components_.shape)
pd.DataFrame(pca.components_.T).loc[:4,:5]
```

```
(11, 11)
```

Out[442]:

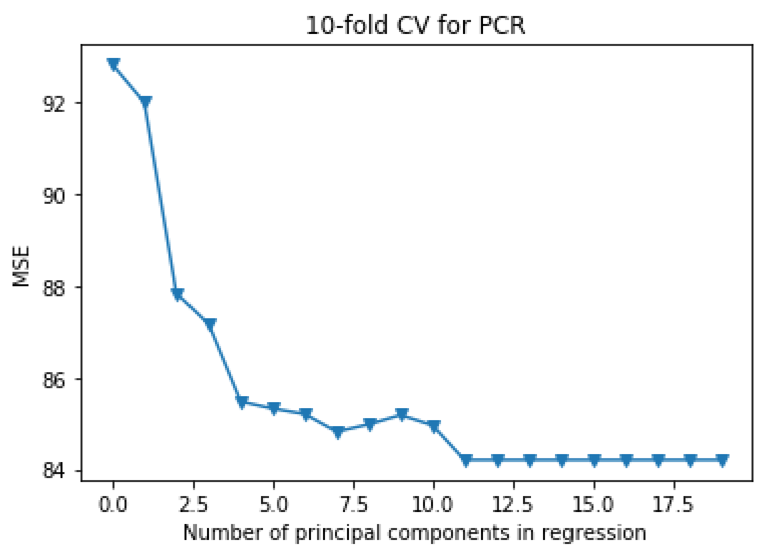
	0	1	2	3	4	5
0	0.213144	-0.574452	0.174492	0.140231	0.083655	0.348872
1	0.333826	0.122690	-0.295611	-0.363472	-0.077992	0.016568
2	0.320097	-0.376640	-0.111123	0.358605	-0.110693	0.122166
3	-0.072662	0.292446	-0.504844	0.445215	0.580374	0.306111
4	-0.284129	-0.361848	-0.365660	-0.200404	0.003681	0.138546

In [443]:
```
# Variance explained by the principal components
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

Out[443]:
```
array([ 24.83,  38.05,  47.83,  56.76,  64.92,  72.32,  78.64,  84.58,
        90.38,  95.45, 100.  ])
```

In [444]:
```
# 10-fold CV
n = len(X_reduced)

regr = LinearRegression()
mse = []

# Calculate MSE with only the intercept (no principal components in regr
ession)
score = -1*cross_val_score(regr, np.ones((n,1)), y_train.ravel(),
                    cv=kf_10, scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using CV for the 19 principle components, adding one com
ponent at the time.
for i in np.arange(1, 20):
    score = -1*cross_val_score(regr, X_reduced[:,:i], y_train.ravel(),
                    cv=kf_10, scoring='neg_mean_squared_error').mean()
    mse.append(score)

plt.plot(mse, '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('10-fold CV for PCR')
plt.xlim(xmin=-1);
mse.index(min(mse))
```

Out[444]: 11



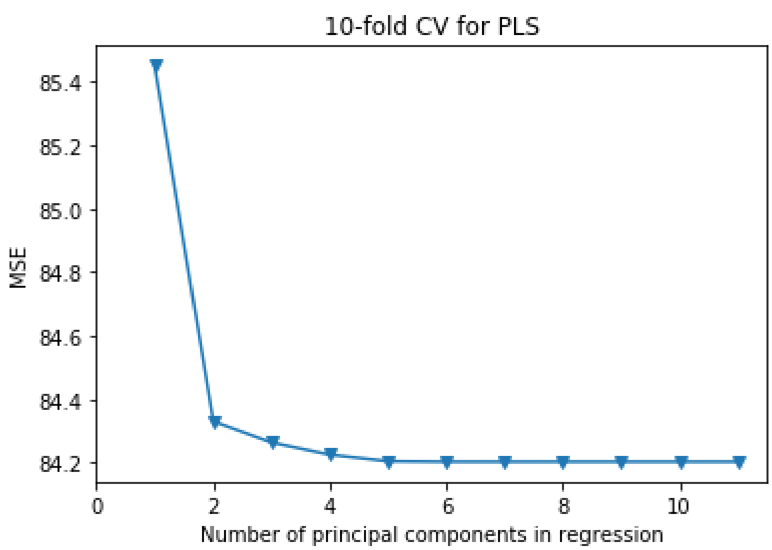
The above block indicates that the lowest training MSE is reached when doing regression on 11 components.

In [449]:
```
pcr = regr.fit(X_reduced[:,:12], y_train)

X_reduced_test = pca.fit_transform(n_test)
pcr_cv_score = -1 * cross_val_score(pcr, X_reduced_test, y_test,
                    cv=kf_10, scoring='neg_mean_squared_error').mean()
cv['Principal component regression'] = pcr_cv_score
```

3d Partial least sqaures regression

In [446]:
```
n = len(X_train)

# 10-fold CV, with shuffle
mse = []

for i in np.arange(1, 12):
    pls = PLSRegression(n_components=i)
    score = cross_val_score(pls, n_train, y_train, cv=kf_10,
                            scoring='neg_mean_squared_error').mean()
    mse.append(-score)

plt.plot(np.arange(1, 12), np.array(mse), '-v')
plt.xlabel('Number of principal components in regression')
plt.ylabel('MSE')
plt.title('10-fold CV for PLS')
plt.xlim(xmin=0);
mse.index(min(mse))
```

Out[446]: 6



In [447]:
```
pls = PLSRegression(n_components=7)
pls.fit(n_train, y_train)
pls_cv_score = -1 * cross_val_score(pls, n_test, y_test,
                    cv=kf_10, scoring='neg_mean_squared_error').mean()
cv['Partial least squares'] = pls_cv_score
```

In [450]:
```
cv
```

Out[450]:
```
{'Linear regression': 86.82184996870981,
 'Elastic net regression': 86.25648563700763,
 'Principal component regression': 86.82184996870981,
 'Partial least squares': 86.82198795408094}
```

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.).

notes from piazza

- You need to explore feature importance by interaction plots, not feature importance plots
- While yes, PCR creates orthogonal components, don't forget how those are created (i.e., via feature-level contributions). Thus, we can still recover some standardize-able (not a word, but you get the idea) version output that is comparable across different models/techniques.
- Notably, you need to first generate predicted values, in order to place all on a single scale first, as you are now directly comparing predicted values, not raw model coefficients, which all vary widely across each other. A way you might think about streamlining the process is to train models with consistent syntax (e.g., via caret), then generate and store predicted values, and then generate feature INXN plots.

In [453]:

```
import eli5
from eli5.sklearn import PermutationImportance
```

In [234]:

```
perm = PermutationImportance(m1, random_state=1).fit(n_test, y_test)
eli5.show_weights(perm, feature_names = X_train.columns.tolist())
```

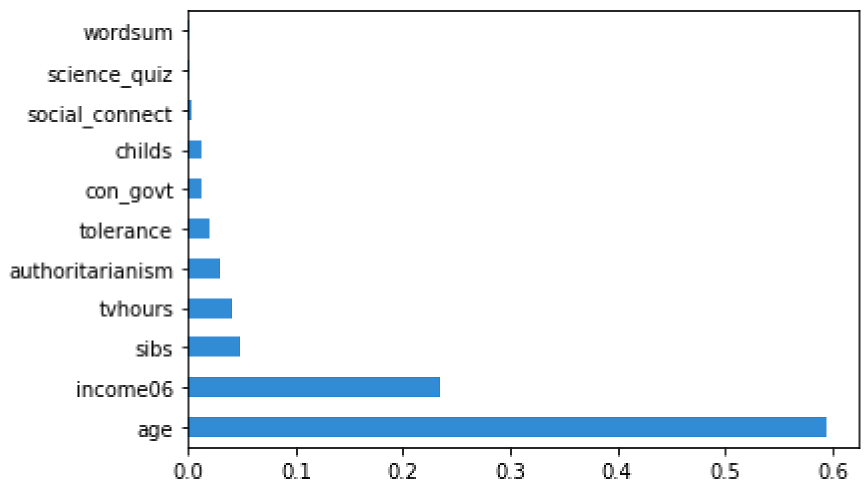
Out[234]:

Weight	Feature
0.0727 ± 0.0201	income06
0.0437 ± 0.0120	age
0.0248 ± 0.0207	tvhours
0.0160 ± 0.0089	con_govt
0.0129 ± 0.0117	sibs
0.0038 ± 0.0106	authoritarianism
0.0006 ± 0.0008	social_connect
-0.0001 ± 0.0004	wordsum
-0.0010 ± 0.0005	science_quiz
-0.0021 ± 0.0059	tolerance
-0.0023 ± 0.0057	childs

```
In [462]: from skater import Interpretation
          interpreter = Interpretation(X_train,
                                       feature_names=X_train.columns)

          from skater.model import InMemoryModel
          mod1 = InMemoryModel(m1.predict, examples = X_train)
          interpreter.load_data(X_train)
          interpreter.feature_importance.plot_feature_importance(mod1,
                                                                 ascending=True)
```

```
[11/11] features  Time elapsed: 0 seconds
```

```
Out[462]: (<Figure size 432x288 with 1 Axes>,
           <matplotlib.axes._subplots.AxesSubplot at 0x1c230b6310>)
```




```
In [235]: perm = PermutationImportance(m2, random_state=1).fit(
                                              n_test, y_test)
          eli5.show_weights(perm, feature_names = X_train.columns.tolist())
```

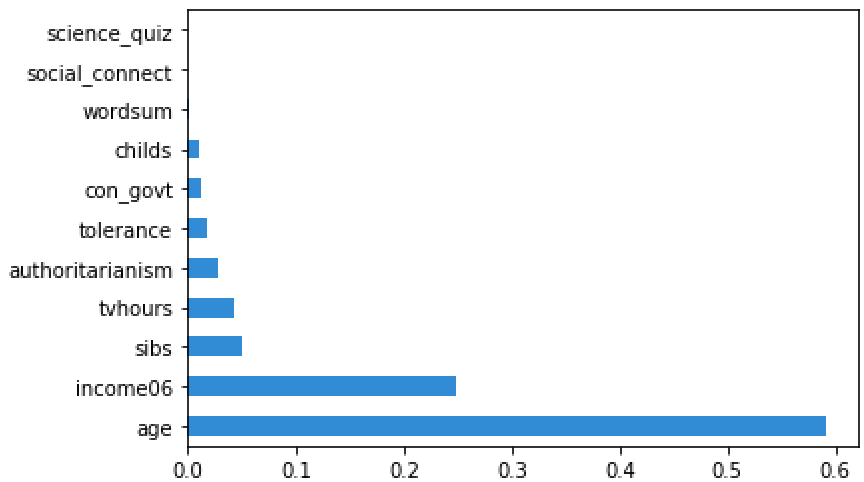
Out[235]:

Weight	Feature
0.0722 ± 0.0196	income06
0.0375 ± 0.0113	age
0.0230 ± 0.0197	tvhours
0.0145 ± 0.0079	con_govt
0.0134 ± 0.0109	sibs
0.0016 ± 0.0089	authoritarianism
0.0001 ± 0.0001	social_connect
-0.0000 ± 0.0000	science_quiz
-0.0001 ± 0.0003	wordsum
-0.0015 ± 0.0046	tolerance
-0.0019 ± 0.0045	childs

In [463]:
```
mod2 = InMemoryModel(m2.predict, examples = X_train)
interpreter.feature_importance.plot_feature_importance(mod2,
                                                        ascending=True)
```

```
[11/11] features  Time elapsed: 0 seconds
```

Out[463]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1c23250d50>)
```



In [451]:
```
perm = PermutationImportance(pcr, random_state=1).fit(
                                        n_test, y_test)
eli5.show_weights(perm, feature_names = X_train.columns.tolist())
```

Out[451]:

Weight	Feature
0.0470 ± 0.0203	con_govt
0.0100 ± 0.0175	authoritarianism
0.0071 ± 0.0200	wordsum
-0.0002 ± 0.0006	social_connect
-0.0030 ± 0.0014	tolerance
-0.0037 ± 0.0132	sibs
-0.0056 ± 0.0204	tvhours
-0.0081 ± 0.0090	age
-0.0188 ± 0.0171	childs
-0.0198 ± 0.0078	science_quiz
-0.0202 ± 0.0063	income06

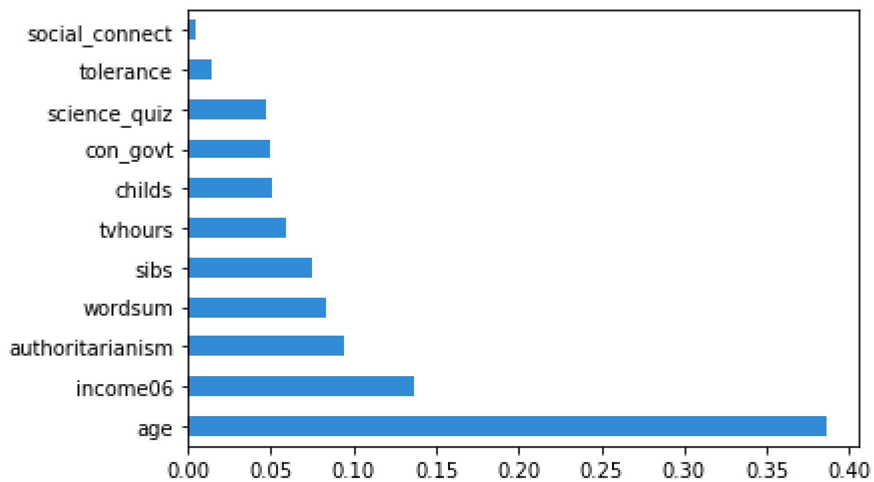
In [464]:

```python
mod3 = InMemoryModel(pcr.predict, examples = X_train)
interpreter.feature_importance.plot_feature_importance(mod3,
                                                       ascending=True)
```

```
[11/11] features  Time elapsed: 0 seconds
```

Out[464]:

```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1c24d33350>)
```



In [237]:

```python
perm = PermutationImportance(pls, random_state=1).fit(n_test, y_test)
eli5.show_weights(perm, feature_names = X_train.columns.tolist())
```

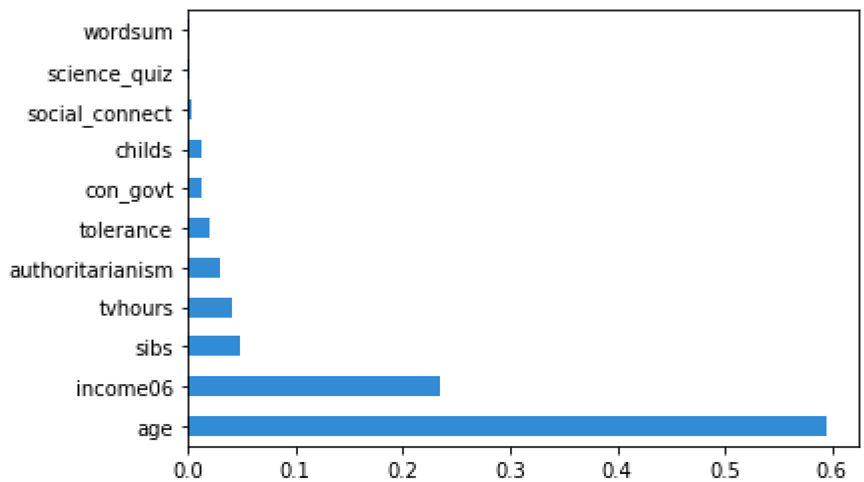
Out[237]:

Weight	Feature
0.0727 ± 0.0201	income06
0.0437 ± 0.0120	age
0.0248 ± 0.0207	tvhours
0.0160 ± 0.0089	con_govt
0.0129 ± 0.0117	sibs
0.0038 ± 0.0106	authoritarianism
0.0006 ± 0.0008	social_connect
-0.0001 ± 0.0004	wordsum
-0.0010 ± 0.0005	science_quiz
-0.0021 ± 0.0059	tolerance
-0.0023 ± 0.0057	childs

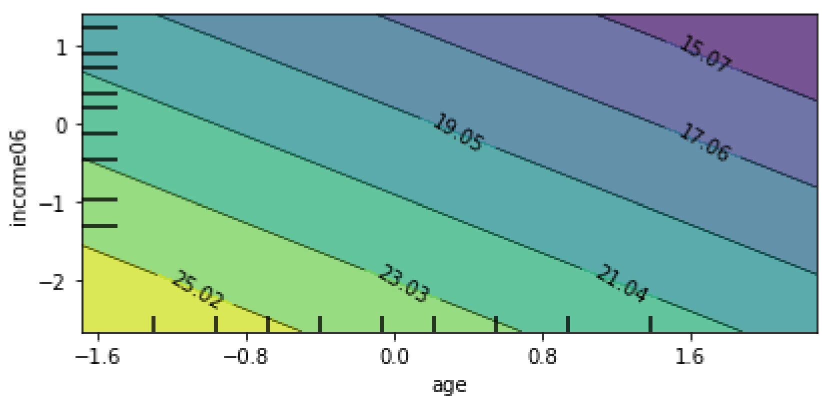
In [465]:
```python
mod4 = InMemoryModel(pls.predict, examples = X_train)
interpreter.feature_importance.plot_feature_importance(mod4,
                                                        ascending=True)
```

```
[11/11] features   Time elapsed: 0 seconds
```

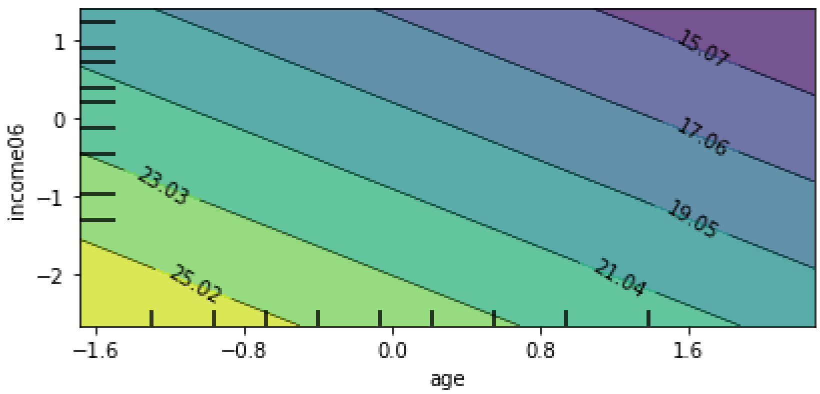
Out[465]:
```
(<Figure size 432x288 with 1 Axes>,
 <matplotlib.axes._subplots.AxesSubplot at 0x1c25419610>)
```



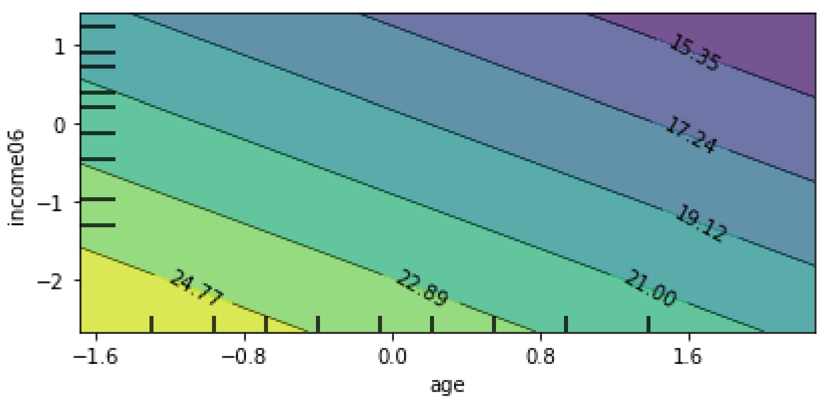
In [473]:
```python
from sklearn.inspection import plot_partial_dependence

f_ls = [(0,4)]
plot_partial_dependence(pls, n_train, features=f_ls,
                        feature_names=X_train.columns)
```



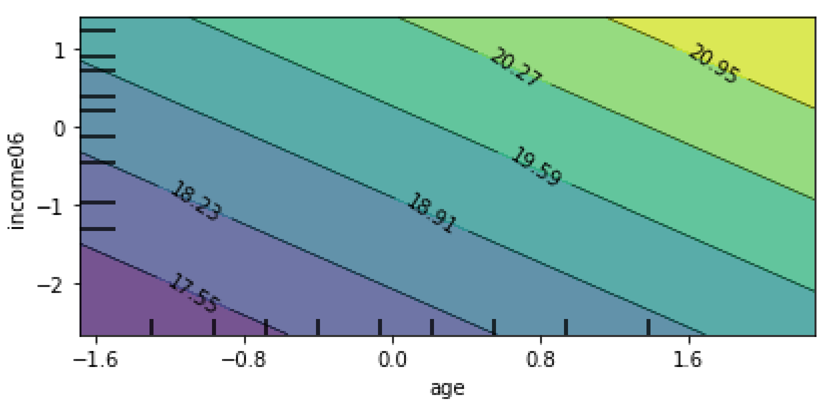
In [474]:
```
plot_partial_dependence(m1, n_train, features=f_ls,
                        feature_names=X_train.columns)
```



In [475]:
```
plot_partial_dependence(m2, n_train, features=f_ls,
                        feature_names=X_train.columns)
```



In [476]:
```
plot_partial_dependence(pcr, n_train, features=f_ls,
                        feature_names=X_train.columns)
```



In [ ]: