HW4

February 16, 2020

[27]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[161]:
```
from sklearn.model_selection import train_test_split, KFold, cross_val_score,␣
 ↪cross_val_predict
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.pipeline import Pipeline
from sklearn.inspection import plot_partial_dependence
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import ElasticNetCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from patsy import dmatrix
import statsmodels.api as sm
import statsmodels.formula.api as smf
```

[5]:
```
gss_test=pd.read_csv("./data/gss_test.csv")
gss_train=pd.read_csv("./data/gss_train.csv")
```

1 Egalitarianism and income

1.1 polynomial regression

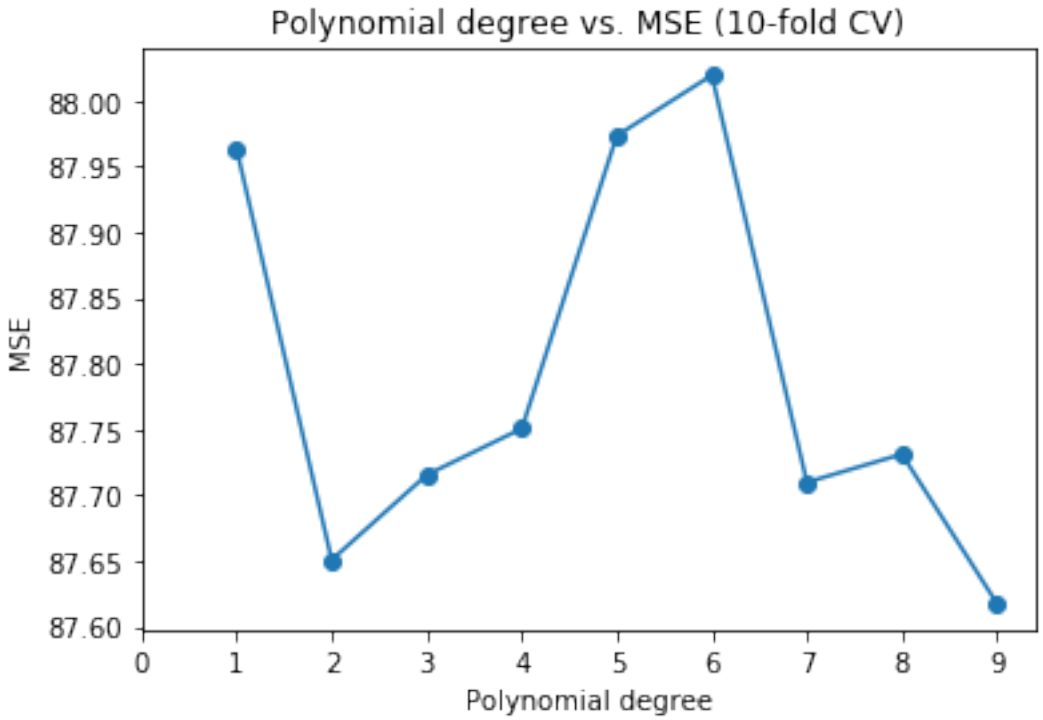
Perform polynomial regression to predict egalit_scale as a function of income06. Use and plot 10-fold cross-validation to select the optimal degree d for the polynomial based on the MSE. Plot the resulting polynomial fit to the data, and also graph the average marginal effect (AME) of income06 across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

[34]:
```
x_train=gss_train['income06'].values.reshape(-1, 1)
y_train=gss_train['egalit_scale']
x_test=gss_test['income06'].values.reshape(-1, 1)
y_test=gss_test['egalit_scale']
```

[37]:
```
# poly model
score_ls=[]
for i in range(1,10):
    poly = Pipeline([('poly', PolynomialFeatures(degree=i)), \
                     ('linear', LinearRegression(fit_intercept=False))])
    poly = poly.fit(x_train, y_train)
    #cross_validation
    scores=cross_val_score(poly, x_train, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
    # average mse
    score_ls.append(np.mean(np.abs(scores)))
```

[41]:
```
# draw mse plot
plt.plot(np.arange(1,10), score_ls, marker='o')
plt.xticks(np.arange(0,10))
plt.xlabel("Polynomial degree")
plt.ylabel("MSE")
plt.title("Polynomial degree vs. MSE (10-fold CV)")
```

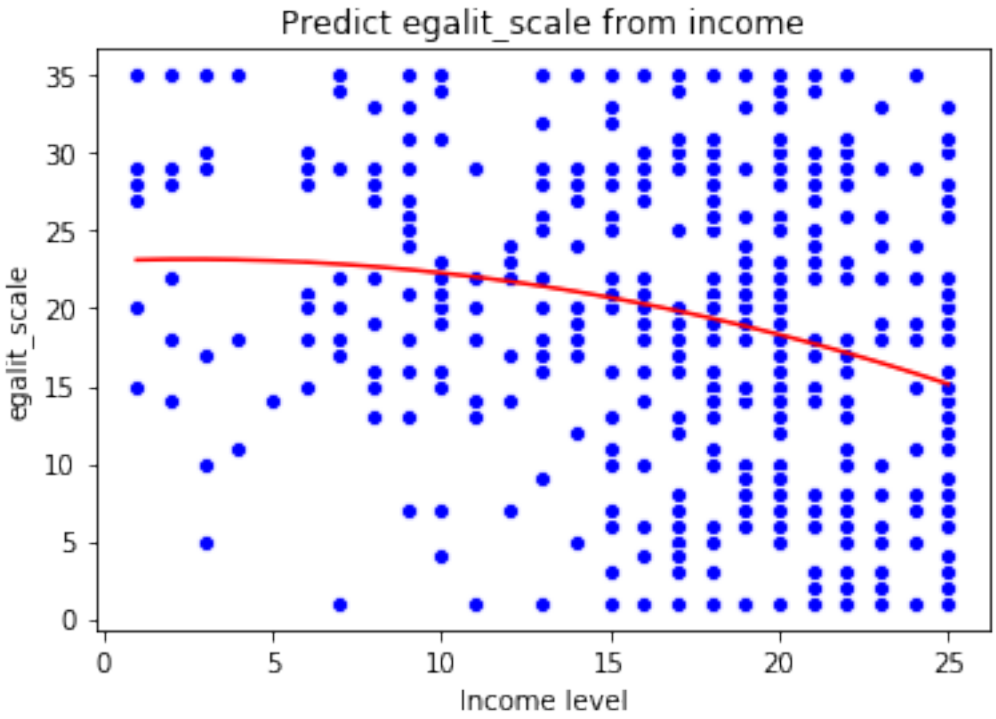
[41]:
```
Text(0.5, 1.0, 'Polynomial degree vs. MSE (10-fold CV)')
```



Based on the above graph, the optimal degree of the polynomial regression model is 2.

[62]:
```
# fit the best polynomial model
poly = Pipeline([('poly', PolynomialFeatures(degree=2)), \
                 ('linear', LinearRegression(fit_intercept=False))])
poly = poly.fit(x_train, y_train)
y_predict=poly.predict(x_test)

sns.scatterplot(x_test.ravel(), y_test,  color='blue')
sns.lineplot(x_test.ravel(), y_predict, color='red')
plt.xlabel("Income level")
plt.title("Predict egalit_scale from income")
```

[62]:
```
Text(0.5, 1.0, 'Predict egalit_scale from income')
```

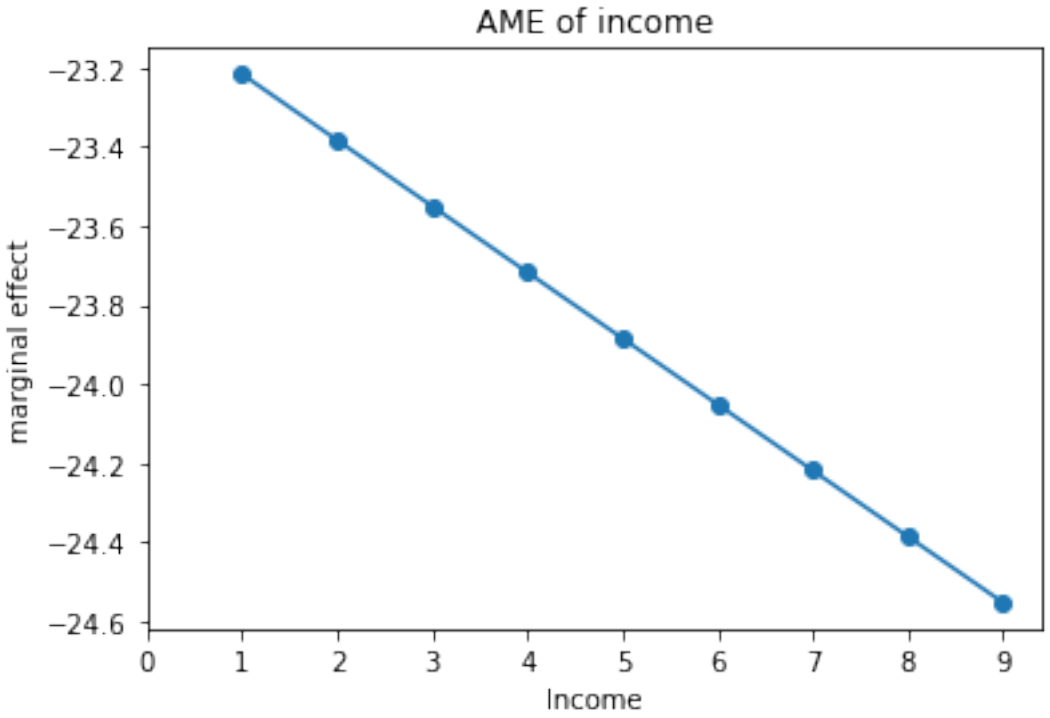


[183]:
```
# marginal effect
def ame_cal(x):
    y=0
    coefs=poly['linear'].coef_
    for i in range(2):
        y+=coefs[i]*(i+1)*(x**(i))
    return y

marg_ls=[]
for x in range(1,10):
```

```
    marg_ls.append(-ame_cal(x))
```

[185]:
```
plt.plot(np.arange(1,10), marg_ls, marker='o')
plt.xticks(np.arange(0,10))
plt.xlabel("Income")
plt.ylabel("marginal effect")
plt.title("AME of income")
```

[185]: Text(0.5, 1.0, 'AME of income')



We found that the optimal degree of the polynomial regression model is 2. Moreover, by the regression graph we could see that the relationship between income and egalit scale follows the trend that higher income leads to lower egalit level. The AME plot shows a negative relationship between the income value and its marginal effect. That said, as income value increase, the marginal effect decreases.

1.2 step function

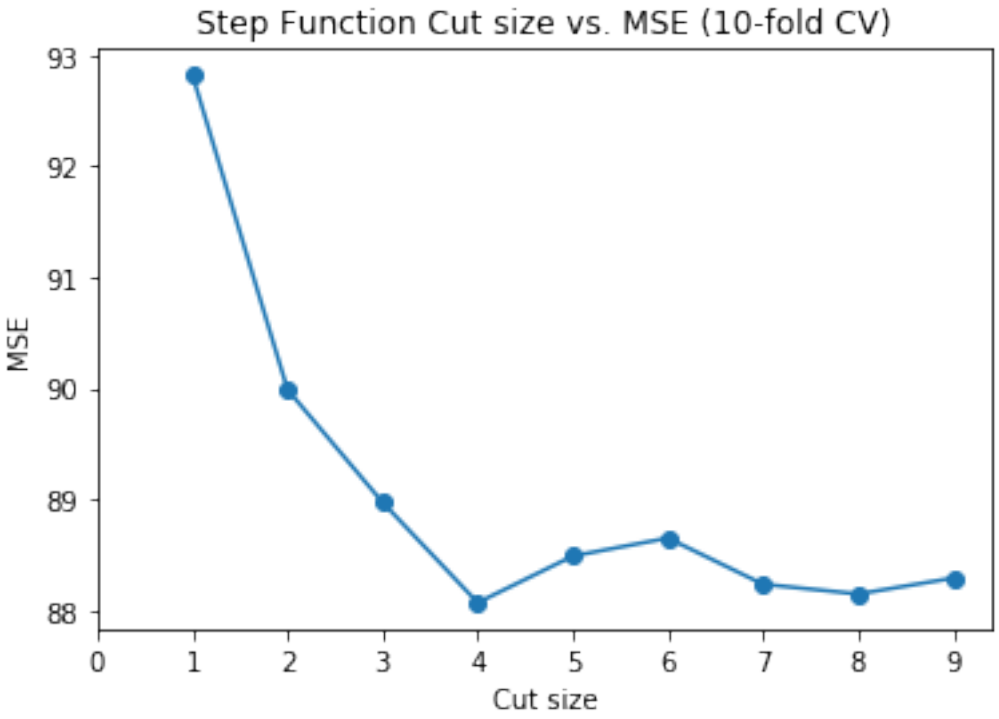
Fit a step function to predict egalit_scale as a function of income06, and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpret the results.

[83]:
```
score_ls=[]

for i in range(1,10):
```

```
    x_train_cut, bins=pd.cut(x_train.ravel(), i, retbins=True)
    step_dummy=pd.get_dummies(x_train_cut)
    step_func=LinearRegression().fit(step_dummy, y_train)
    scores=cross_val_score(step_func, step_dummy, y_train,␣
↪scoring="neg_mean_squared_error", cv=10)
    score_ls.append(np.mean(np.abs(scores)))
```

[84]:
```
# draw mse plot
plt.plot(np.arange(1,10), score_ls, marker='o')
plt.xticks(np.arange(0,10))
plt.xlabel("Cut size")
plt.ylabel("MSE")
plt.title("Step Function Cut size vs. MSE (10-fold CV)")
```

[84]:
```
Text(0.5, 1.0, 'Step Function Cut size vs. MSE (10-fold CV)')
```

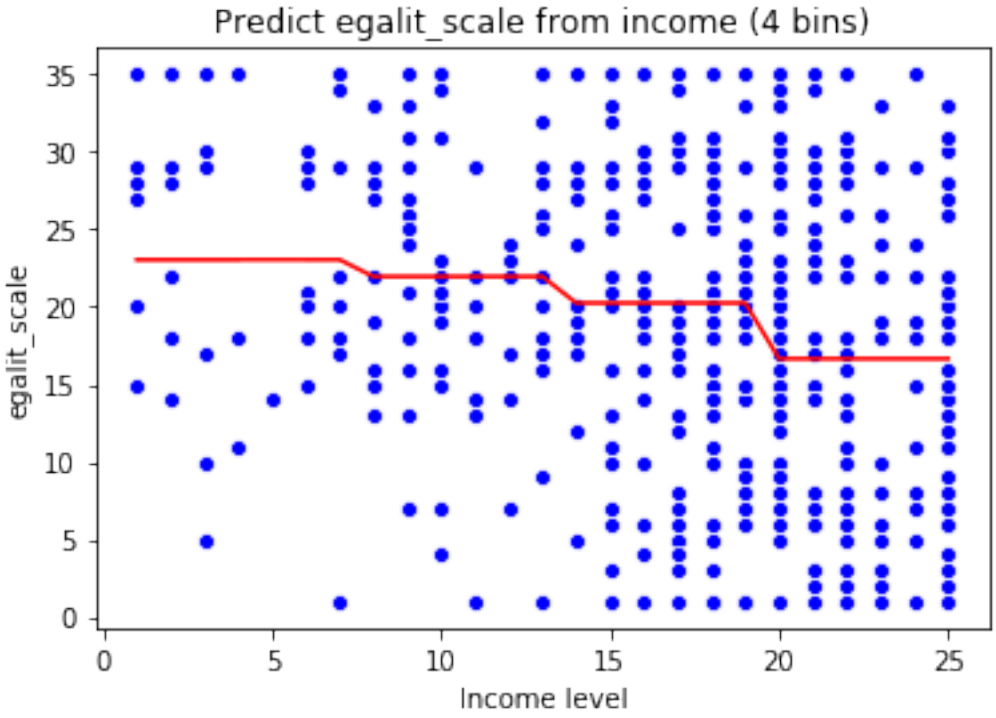


From the graph we could see that the optimal cut size is 4.

[89]:
```
# fit the best step model
x_train_cut, bins=pd.cut(x_train.ravel(), 4, retbins=True)
step_dummy=pd.get_dummies(x_train_cut)
step_func=LinearRegression().fit(step_dummy, y_train)

x_test_cut, bins=pd.cut(x_test.ravel(), 4, retbins=True)
```

```
step_dummy_test=pd.get_dummies(x_test_cut)
y_predict=step_func.predict(step_dummy_test)

sns.scatterplot(x_test.ravel(), y_test,  color='blue')
sns.lineplot(x_test.ravel(), y_predict, color='red')
plt.xlabel("Income level")
plt.title("Predict egalit_scale from income (4 bins)")
```

[89]: Text(0.5, 1.0, 'Predict egalit_scale from income (4 bins)')



From the graph, we could see that the relationship between income level and egalit scale follows similar trend as polynomial regression. As income increases, the egalit scale drops slightly. Note that compare to polynomial regression, the curve of the step function is very rigid.

1.3 natural regression

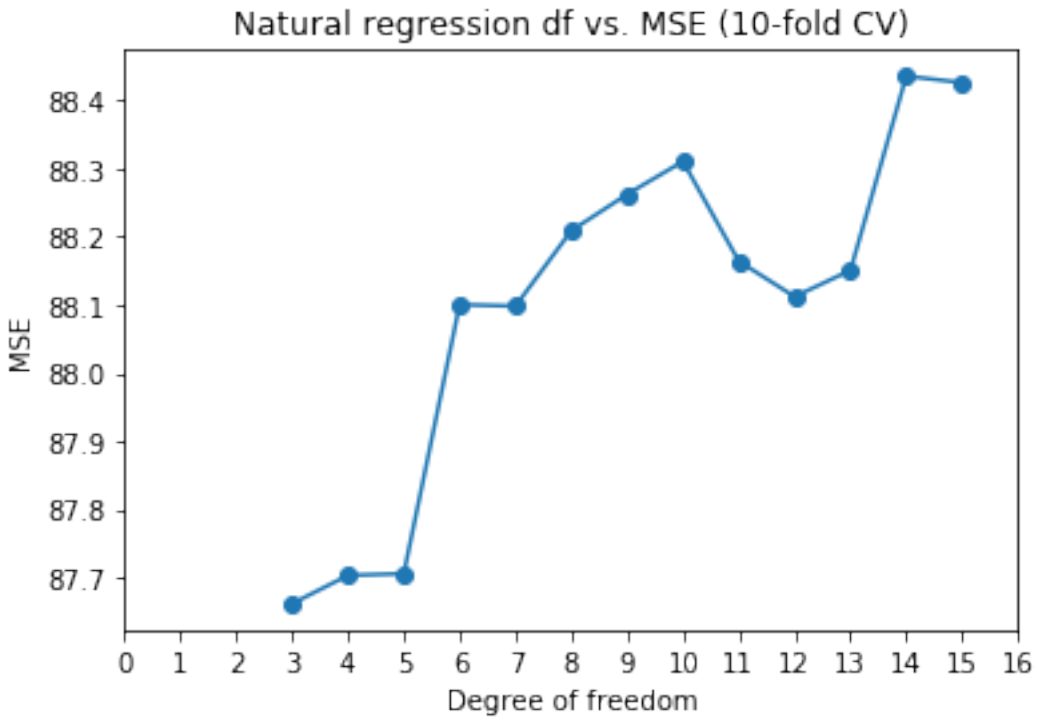
Fit a natural regression spline to predict egalit_scale as a function of income06. Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the results of the optimal model.

Reference: https://www.analyticsvidhya.com/blog/2018/03/introduction-regression-splines-python-codes/

[106]:
```
score_ls=[]
for i in range(3,16):
    transformed_x = dmatrix("cr(train,df)", {"train": x_train, "df":i},␣
 ↪return_type='dataframe')
    natural_reg=LinearRegression().fit(transformed_x, y_train)
    scores=cross_val_score(natural_reg, transformed_x, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
    score_ls.append(np.mean(np.abs(scores)))
```

[109]:
```
# draw mse plot
plt.plot(np.arange(3,16), score_ls, marker='o')
plt.xticks(np.arange(0,17))
plt.xlabel("Degree of freedom")
plt.ylabel("MSE")
plt.title("Natural regression df vs. MSE (10-fold CV)")
```

[109]:
```
Text(0.5, 1.0, 'Natural regression df vs. MSE (10-fold CV)')
```

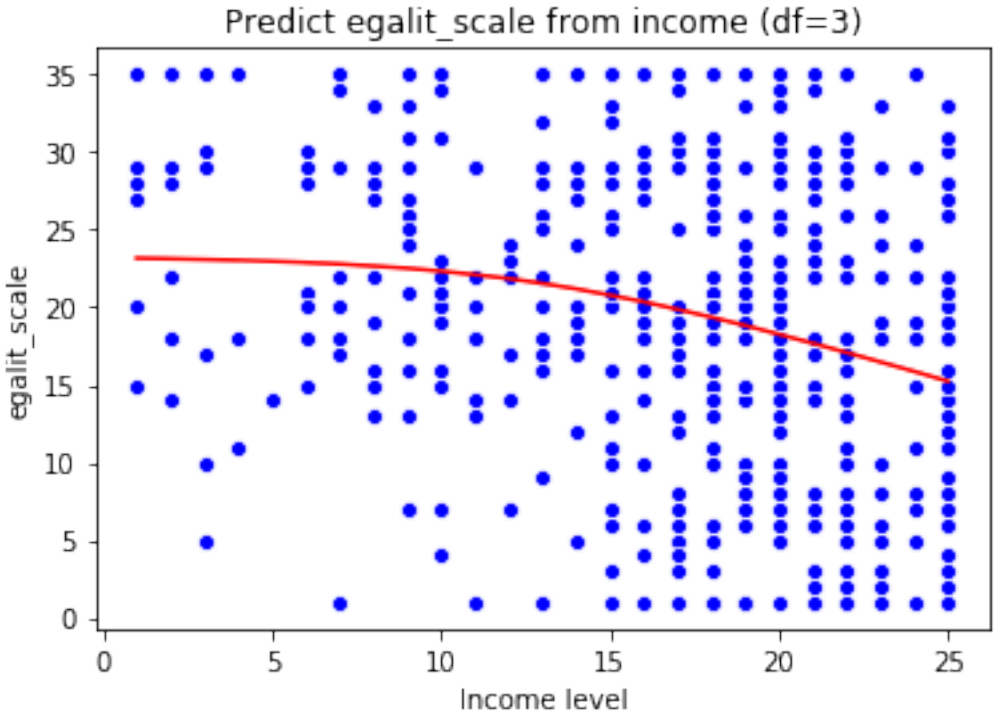


From the plot, we could see that the optimal df is 3.

[110]:
```
# fit the best step model
transformed_x = dmatrix("cr(train,df)", {"train": x_train, "df":3},␣
 ↪return_type='dataframe')
natural_reg=LinearRegression().fit(transformed_x, y_train)
```

```
transformed_x_test = dmatrix("cr(test,df)", {"test": x_test, "df":3},␣
 ↪return_type='dataframe')
y_predict=natural_reg.predict(transformed_x_test)

sns.scatterplot(x_test.ravel(), y_test,  color='blue')
sns.lineplot(x_test.ravel(), y_predict, color='red')
plt.xlabel("Income level")
plt.title("Predict egalit_scale from income (df=3)")
```

[110]: `Text(0.5, 1.0, 'Predict egalit_scale from income (df=3)')`



The above picture shows the fitting result using natural spline regression to predict egalit scale based on income level. The curve of natural spline shows similar trend compared to step function and polynomial function. Moreover, it looks very similar to that of polynomial function. Since the degree of freedome here is only 3, such similarity is understandable.

2 Egalitarianism and everything

2.1 Extra model estimation

Estimate the following models using all the available predictors

[146]:
```
x_train=gss_train.drop("egalit_scale", axis=1)
x_test=gss_test.drop("egalit_scale", axis=1)
```

```
y_train=gss_train["egalit_scale"]
y_test=gss_test["egalit_scale"]
```

[147]:
```
# feature standardization
def feature_std(df_train, df_test):
    for col in df_train:
        if df_train[col].dtypes==object:
            enc=LabelEncoder()
            df_train[col]=enc.fit_transform(df_train[col])
            df_test[col]=enc.transform(df_test[col])
        else:
            std_scaler_tr=StandardScaler()
            train_col=df_train[col].values.reshape(-1, 1)
            std_scaler_tr=std_scaler_tr.fit(train_col)
            df_train[col]=std_scaler_tr.transform(train_col)

            std_scaler_ts=StandardScaler()
            test_col=df_test[col].values.reshape(-1, 1)
            std_scaler_ts=std_scaler_ts.fit(test_col)
            df_test[col]=std_scaler_ts.transform(test_col)

    return df_train, df_test
```

[148]:
```
x_train_std, x_test_std=feature_std(x_train, x_test)
```

2.1.1 Linear regression

[155]:
```
lm=GridSearchCV(LinearRegression(), {}, scoring="neg_mean_squared_error", cv=10)
lm.fit(x_train_std, y_train)
best_lm=lm.best_estimator_
```

[156]:
```
scores=cross_val_score(best_lm, x_train_std, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
lm_train_mse=np.mean(np.abs(scores))

y_predict=best_lm.predict(x_test_std)
lm_test_mse=mean_squared_error(y_test, y_predict)

print(f"The Test MSE for linear regression is {lm_test_mse}")
print(f"The Train MSE for linear regression is {lm_train_mse}")
```

```
The Test MSE for linear regression is 63.88809410027567
The Train MSE for linear regression is 64.73968280383566
```

2.1.2 Elastic net regression

[151]:
```
enet=ElasticNetCV([.1, .5, .7, .9, .95, .99, 1], cv=10).fit(x_train_std,␣
↪y_train)
scores=cross_val_score(enet, x_train_std, y_train,␣
↪scoring="neg_mean_squared_error", cv=10)
enet_train_mse=np.mean(np.abs(scores))

y_predict=enet.predict(x_test_std)
enet_test_mse=mean_squared_error(y_test, y_predict)

print(f"The Test MSE for Elastic Net is {enet_test_mse}")
print(f"The Train MSE for Elastic Net is {enet_train_mse}")
```

```
The Test MSE for Elastic Net is 62.88597874051978
The Train MSE for Elastic Net is 63.62474520308994
```

2.1.3 Principal component regression

[162]:
```
pca=Pipeline([('pca',PCA()),('lm',LinearRegression())])
param_grid={'pca__n_components':np.arange(1,30)}
pca_cv=GridSearchCV(pca, param_grid, cv=10)
pca_cv.fit(x_train_std, y_train)
best_pca=pca_cv.best_estimator_
```

```
/Users/ziwenchen/opt/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:814: DeprecationWarning: The default
of the `iid` parameter will change from True to False in version 0.22 and will
be removed in 0.24. This will change numeric results when test-set sizes are
unequal.
  DeprecationWarning)
```

[163]:
```
scores=cross_val_score(best_pca, x_train_std, y_train,␣
↪scoring="neg_mean_squared_error", cv=10)
pca_train_mse=np.mean(np.abs(scores))

y_predict=best_pca.predict(x_test_std)
pca_test_mse=mean_squared_error(y_test, y_predict)

print(f"The Test MSE for linear regression is {pca_test_mse}")
print(f"The Train MSE for linear regression is {pca_train_mse}")
```

```
The Test MSE for linear regression is 63.54328930460201
The Train MSE for linear regression is 65.63392542666672
```

2.1.4 Partial least squares regression

[164]:
```
param_grid={'n_components':np.arange(1,30)}
pcr_cv=GridSearchCV(PLSRegression(),param_grid,cv=10)
pcr_cv.fit(x_train_std, y_train)
best_pcr=pcr_cv.best_estimator_
```

[165]:
```
scores=cross_val_score(best_pcr, x_train_std, y_train,␣
 ↪scoring="neg_mean_squared_error", cv=10)
pcr_train_mse=np.mean(np.abs(scores))

y_predict=best_pcr.predict(x_test_std)
pcr_test_mse=mean_squared_error(y_test, y_predict)

print(f"The Test MSE for linear regression is {pcr_test_mse}")
print(f"The Train MSE for linear regression is {pcr_train_mse}")
```

```
The Test MSE for linear regression is 61.564618588499144
The Train MSE for linear regression is 64.75709231238716
```

2.2 Evaluation

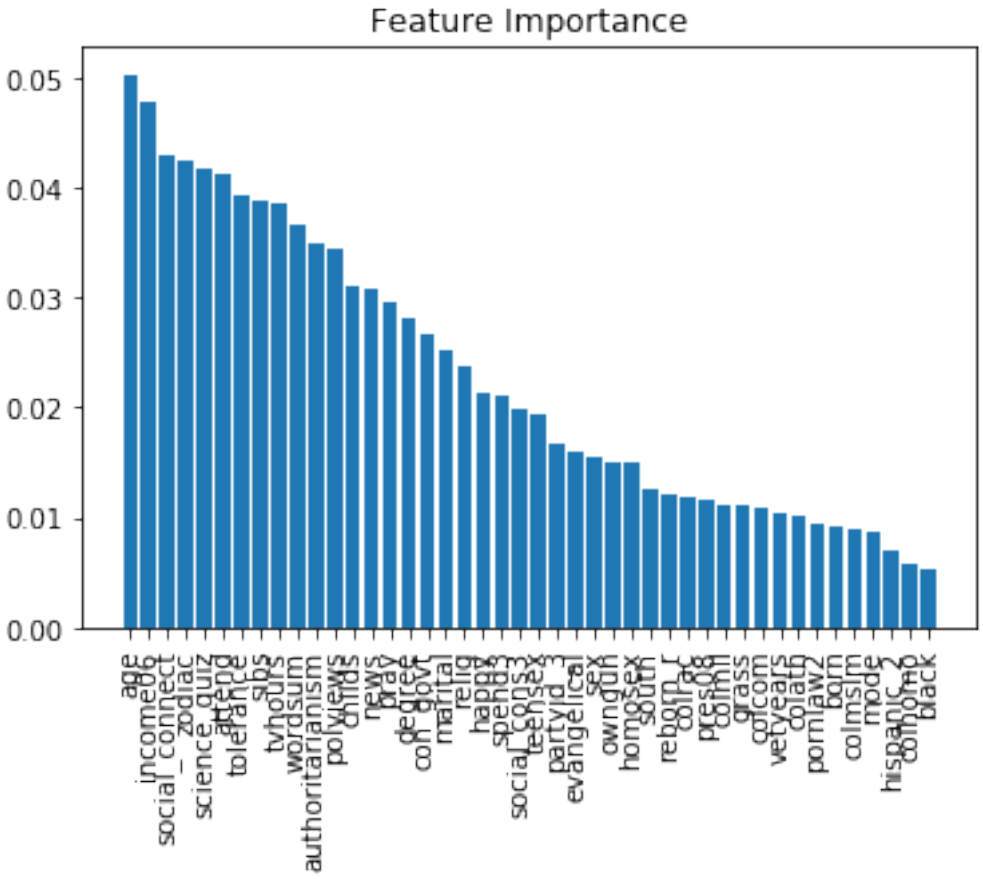
For each final tuned version of each model fit, evaluate feature importance by generating feature interaction plots. Upon visual presentation, be sure to discuss the substantive results for these models and in comparison to each other

[167]:
```
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(random_state=0, n_jobs=-1)
model=clf.fit(x_train_std, y_train)
```

```
/Users/ziwenchen/opt/anaconda3/lib/python3.7/site-
packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of
n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

[172]:
```
# Calculate feature importances
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

# Rearrange feature names so they match the sorted feature importances
names = [x_train_std.columns[i] for i in indices]

# Create plot
plt.figure()

# Create plot title
plt.title("Feature Importance")
```

```
# Add bars
plt.bar(range(x_train_std.shape[1]), importances[indices])

# Add feature names as x-axis labels
plt.xticks(range(x_train_std.shape[1]), names, rotation=90)

# Show plot
plt.show()
```



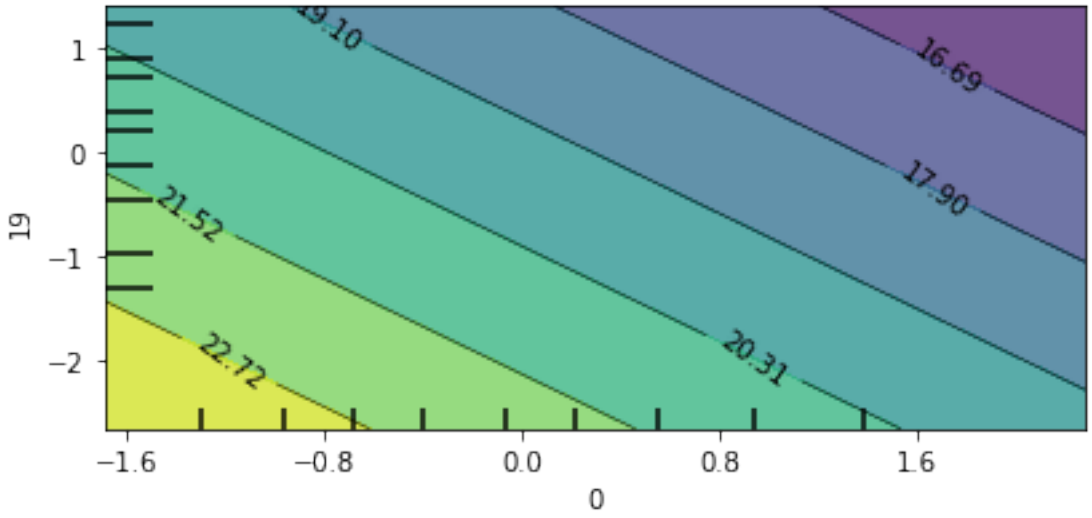
[176]:
```
col_ls=list(x_train_std.columns)
print(col_ls.index('age'))
print(col_ls.index('income06'))
```

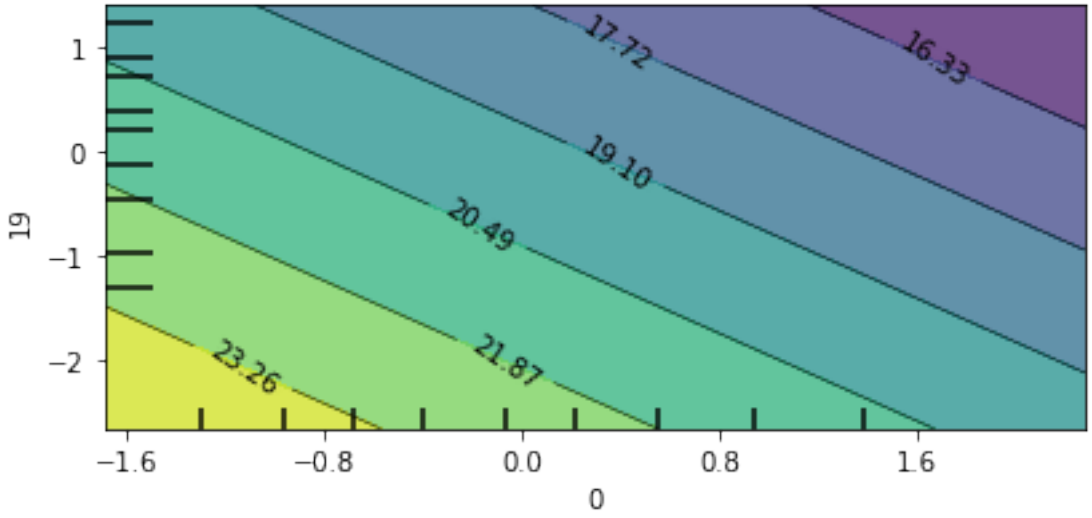
```
0
19
```

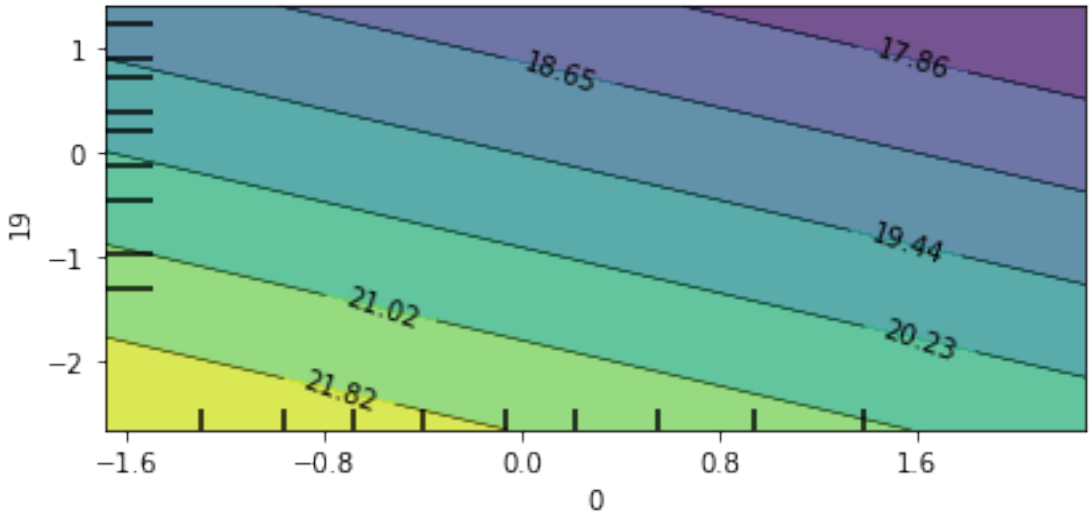
We therefore select age, income to draw interaction plot.

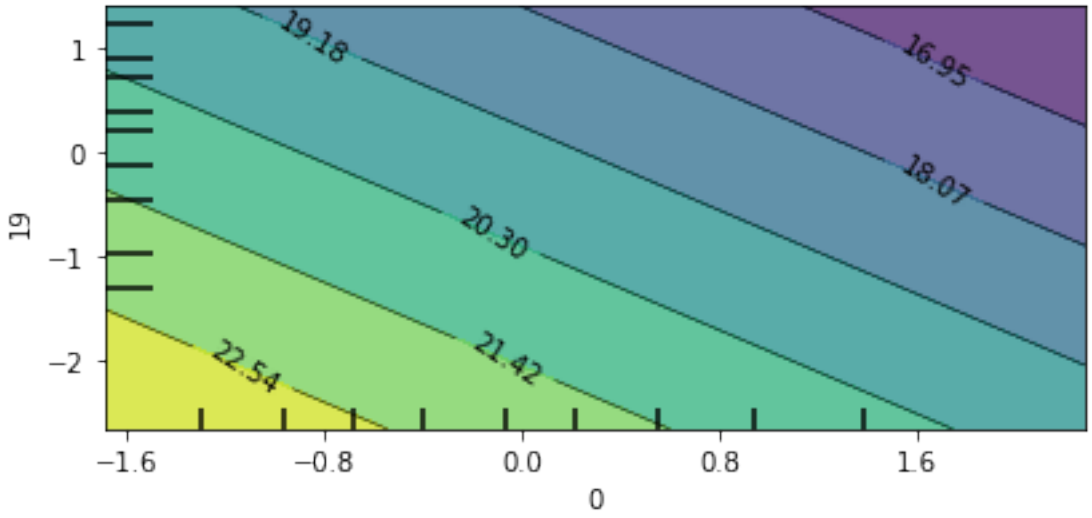
[178]:
```
# linear
plot_partial_dependence(best_lm, x_train_std, [(0,19)])
```

```
plot_partial_dependence(best_pca, x_train_std, [(0,19)])
plot_partial_dependence(best_pcr, x_train_std, [(0,19)])
plot_partial_dependence(enet, x_train_std, [(0,19)])
```









I plot the feature interaction map between income and age for the four different models. Generally,the way these two features interact is very similar to each other. Among all the four methods, we could see that PCR has the largest difference compared to others. Since age and income are the most "influential" features, and we also observed that all models have similar performance in predicting task, such pattern is understandable.