Homework 4

February 17, 2020

0.1 Homework 4-Yutao Chen

```
In [1]: import numpy as np
        import scipy as sp
        import matplotlib.pyplot as plt
        from pylab import rcParams
        import seaborn as sns
        import pandas as pd
        import sklearn
        import warnings
        import itertools
        import sklearn.metrics
        from sklearn.linear_model import LinearRegression
        warnings.filterwarnings('ignore')
```

0.1.1 Egalitarianism and income

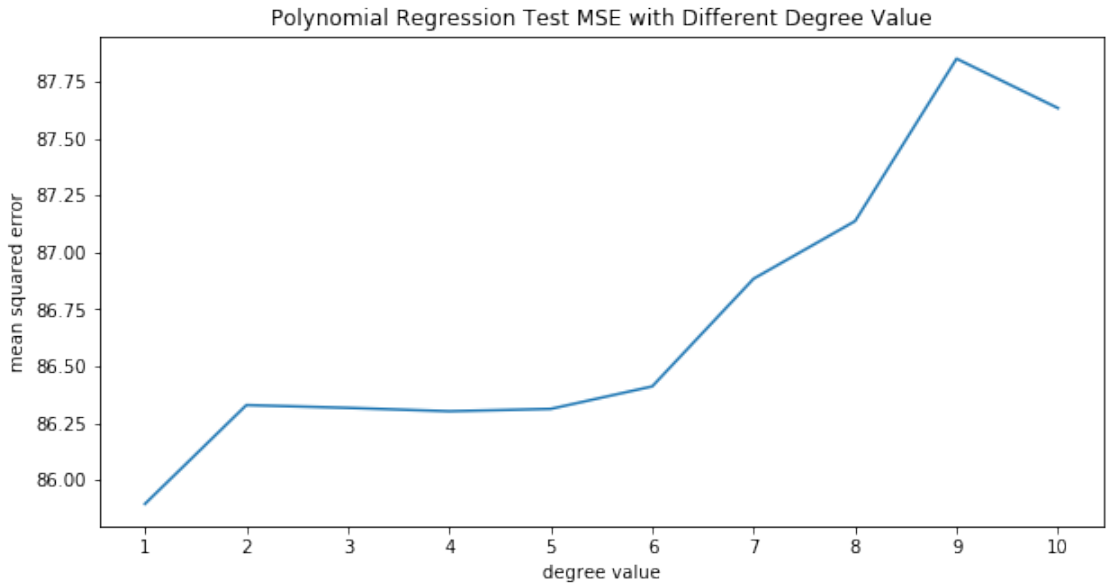
```
In [2]: # loading data
        gss_train = pd.read_csv("data/gss_train.csv")
        gss_test = pd.read_csv("data/gss_test.csv")
```

Q1 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 [89]: import sklearn.preprocessing
         import sklearn.linear_model

         polynomial_result = []
         for d in range(1,11):
             neg_mse = sklearn.metrics.make_scorer(
             sklearn.metrics.mean_squared_error, greater_is_better = False)

             # building the model for each degree
             poly_converter = sklearn.preprocessing.PolynomialFeatures(degree=d)
             poly_features = poly_converter.fit_transform(
             np.array(gss_train['income06']).reshape(-1, 1))
```

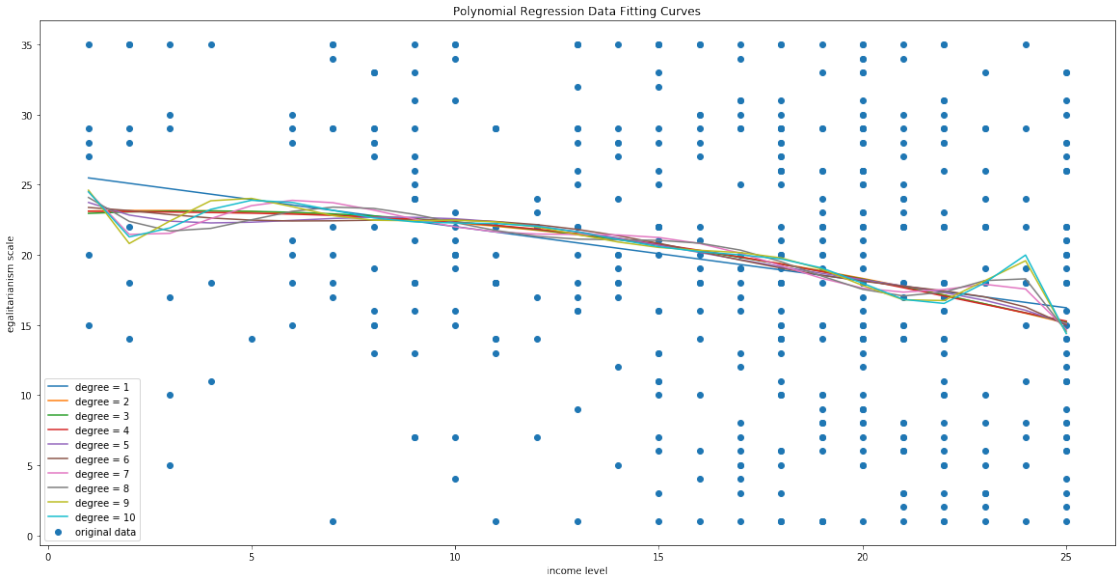
```
        model = sklearn.linear_model.RidgeCV(alphas=[0.0],
        normalize=True, scoring=neg_mse, cv=10)

        # fitting model
        model.fit(poly_features, gss_train['egalit_scale'])
        test_predict = model.predict(
        poly_converter.fit_transform(
        np.array(gss_test['income06']).reshape(-1, 1)))
        test_mse = sklearn.metrics.mean_squared_error(
        gss_test['egalit_scale'], test_predict)

        polynomial_result.append((d,test_predict,test_mse,model))
```

In [91]:
```
# plotting the mse variance with different degree d
plt.rcParams["figure.figsize"] = (10,5)
plt.plot([x[0] for x in polynomial_result],
         [x[2] for x in polynomial_result])
plt.title('Polynomial Regression Test MSE with Different Degree Value')
plt.xticks(range(1,11))
plt.xlabel('degree value')
plt.ylabel('mean squared error')
#plt.legend()
plt.show()
```



In [92]:
```
# plotting data fitting curves of different models
plt.rcParams["figure.figsize"] = (20,10)
plt.scatter(gss_test['income06'],
            gss_test['egalit_scale'],
```

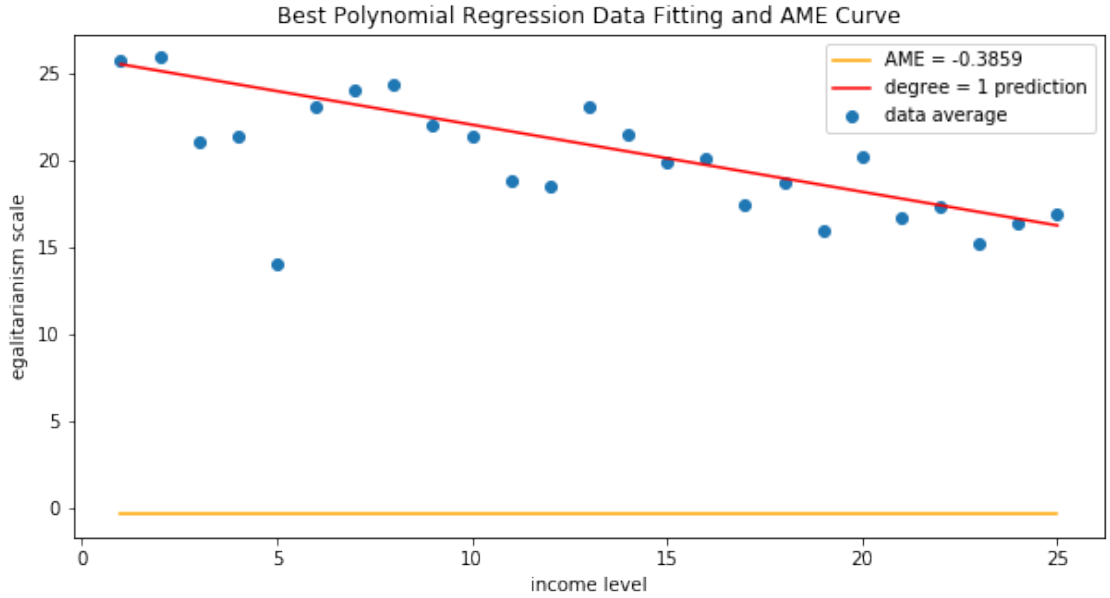
```
                label='original data' )

    for item in polynomial_result:
        poly_converter = sklearn.preprocessing.PolynomialFeatures(degree=item[0])
        x = np.unique(gss_test['income06']).reshape(-1, 1)
        poly_features = poly_converter.fit_transform(x)
        model = item[3]
        plt.plot(x, model.predict(poly_features),
        label = 'degree = {}'.format(item[0]))

    plt.title('Polynomial Regression Data Fitting Curves')
    plt.xlabel('income level')
    plt.ylabel('egalitarianism scale')
    plt.legend()
    plt.show()
```



According to the test MSE calculated, we find that the optimal degree is 1, which means that the model is the best when is linear, so the AME in this situation should be constant.

```
In [95]: # plotting the average marginal effect (AME) of income06 and the best fitting
        plt.rcParams["figure.figsize"] = (10,5)
        plt.scatter(gss_test.groupby(['income06']).mean()['egalit_scale'].index,
                    gss_test.groupby(['income06']).mean()['egalit_scale'],
                    label='data average' )

        Best_Poly = sorted(polynomial_result, key= lambda x:x[2])[0]
        Best_Poly_Model = Best_Poly[3]
        poly_converter = sklearn.preprocessing.PolynomialFeatures(degree=Best_Poly[0])
        x = np.unique(gss_test['income06']).reshape(-1, 1)
```

```
plt.plot(x, [Best_Poly_Model.coef_[1]]*len(x),
label = 'AME = {}'.format(round(Best_Poly_Model.coef_[1],4)),color ='orange')
plt.plot(x, Best_Poly_Model.predict(poly_converter.fit_transform(x)),
label = 'degree = {} prediction'.format(Best_Poly[0]),color = 'red')

plt.title('Best Polynomial Regression Data Fitting and AME Curve')
plt.xlabel('income level')
plt.ylabel('egalitarianism scale')
plt.legend()
plt.show()
```



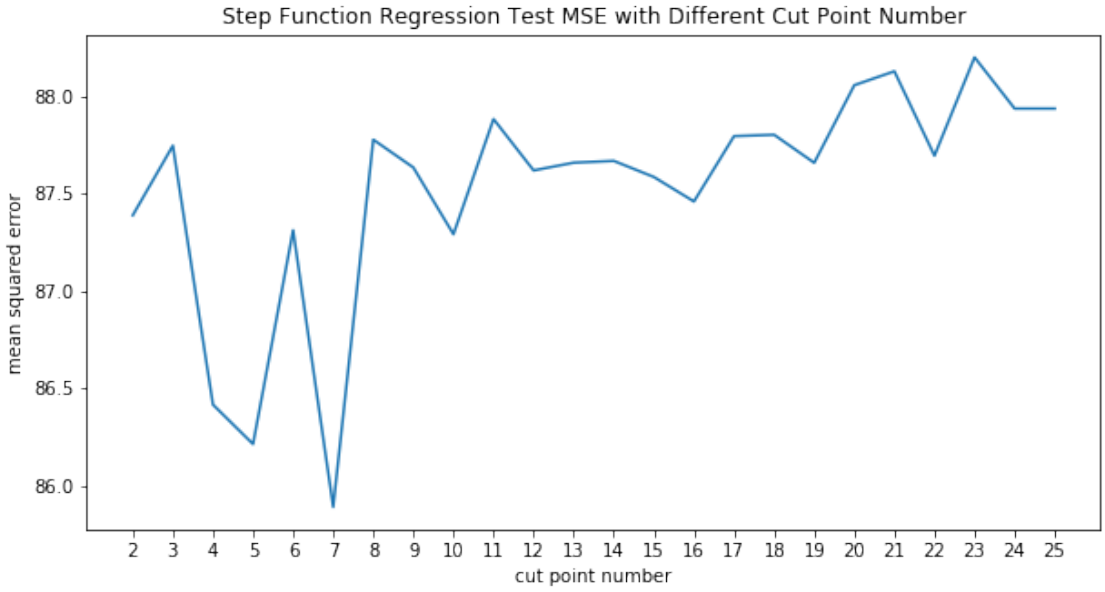
From the plot above, we can find that the optimal polynomial regression results show that the relationship between income and egalitarianism is more likely to be linear (degree = 1). Also, the correlation between the two variables tends to be negative, which means that a higher level of total family income is more likely to be associated with a relatively lower egalitarianism scale.

Q2 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 [128]: step_result = []
          all_income =list(gss_train['income06']) + list(gss_test['income06'])
          x = np.unique(all_income).reshape(-1, 1)
          for i in range(2,len(x)+1):
              # constructing bins based on quantile
              bins_converter = sklearn.preprocessing.KBinsDiscretizer(
                  n_bins=i, encode='onehot-dense')
              train_onehot_features = bins_converter.fit_transform(
```

```
        np.array(all_income).reshape(-1, 1))[:len(gss_train)]

    # building cross-validation regression model
    neg_mse = sklearn.metrics.make_scorer(
        sklearn.metrics.mean_squared_error, greater_is_better = False)
    model = sklearn.linear_model.RidgeCV(
        alphas=[0.0], scoring=neg_mse, normalize=True, cv=10)
    model.fit(train_onehot_features, gss_train['egalit_scale'])

    # prediction and testing
    test_onehot_features = bins_converter.fit_transform(
        np.array(all_income).reshape(-1, 1))[len(gss_train):]
    test_predict = model.predict(test_onehot_features)
    test_mse = sklearn.metrics.mean_squared_error(
    gss_test['egalit_scale'], test_predict)

    step_result.append((i, test_mse, model))
```

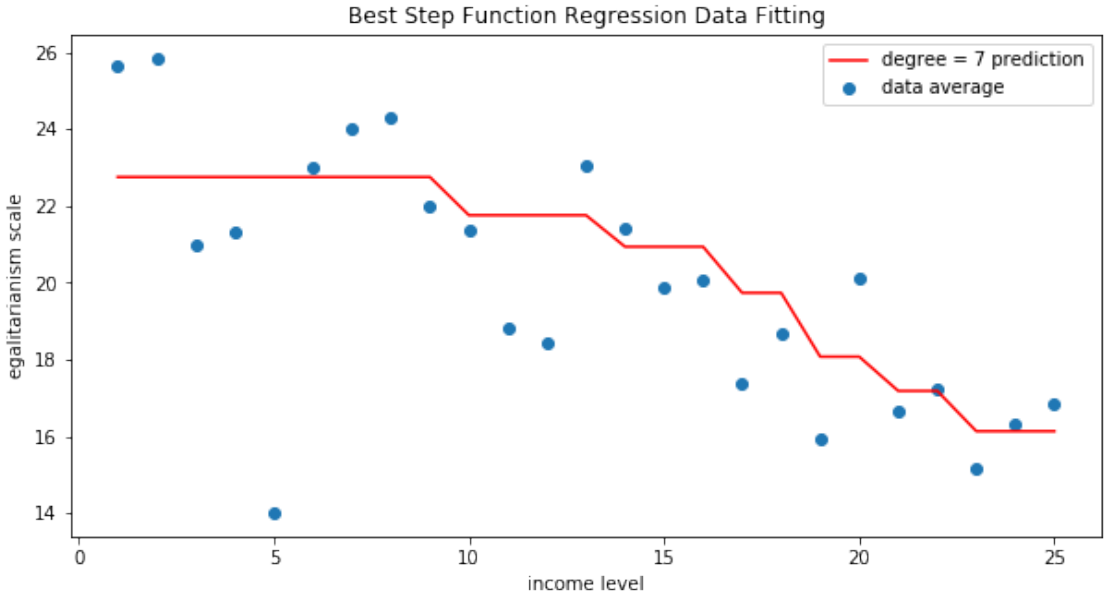
In [130]:
```
# plotting the mse variance with different cut points
plt.rcParams["figure.figsize"] = (10,5)
plt.plot([x[0] for x in step_result],
         [x[1] for x in step_result])
plt.title('Step Function Regression Test MSE with Different Cut Point Number')
plt.xticks(range(2,len(x)+1))
plt.xlabel('cut point number')
plt.ylabel('mean squared error')
plt.show()
```



In [132]:
```
# plotting data fitting curves of different models
plt.rcParams["figure.figsize"] = (20,10)
plt.scatter(gss_test['income06'],
            gss_test['egalit_scale'],
            label='original data' )

for item in step_result:
    bins_converter = sklearn.preprocessing.KBinsDiscretizer(
    n_bins=item[0], encode='onehot-dense')
    x = np.unique(gss_test['income06']).reshape(-1, 1)
    bins_converter.fit(np.array(all_income).reshape(-1, 1))
    onehot_features = bins_converter.transform(x)
    model = item[2]
    plt.plot(x, model.predict(onehot_features),
    label = 'cut number = {}'.format(item[0]))

plt.title('Step Function Regression Data Fitting Curves')
plt.xlabel('income level')
plt.ylabel('egalitarianism scale')
plt.legend()
plt.show()
```

In [134]:
```
# plotting the best fitting
plt.rcParams["figure.figsize"] = (10,5)
plt.scatter(gss_test.groupby(['income06']).mean()['egalit_scale'].index,
            gss_test.groupby(['income06']).mean()['egalit_scale'],
            label='data average' )
```

```
Best_Step = sorted(step_result, key= lambda x:x[1])[0]
Best_Step_Model = Best_Step[2]
bins_converter = sklearn.preprocessing.KBinsDiscretizer(
n_bins=Best_Step[0], encode='onehot-dense')
x = np.unique(gss_test['income06']).reshape(-1, 1)
bins_converter.fit(np.array(all_income).reshape(-1, 1))
plt.plot(x, Best_Step_Model.predict(bins_converter.transform(x)),
label = 'degree = {} prediction'.format(Best_Step[0]), color = 'red')

plt.title('Best Step Function Regression Data Fitting')
plt.xlabel('income level')
plt.ylabel('egalitarianism scale')
plt.legend()
plt.show()
```




```
In [136]: print("Best polynomial regression MSE: {}".format(Best_Poly[2]))
          print("Best step function regression MSE: {}".format(Best_Step[1]))
```

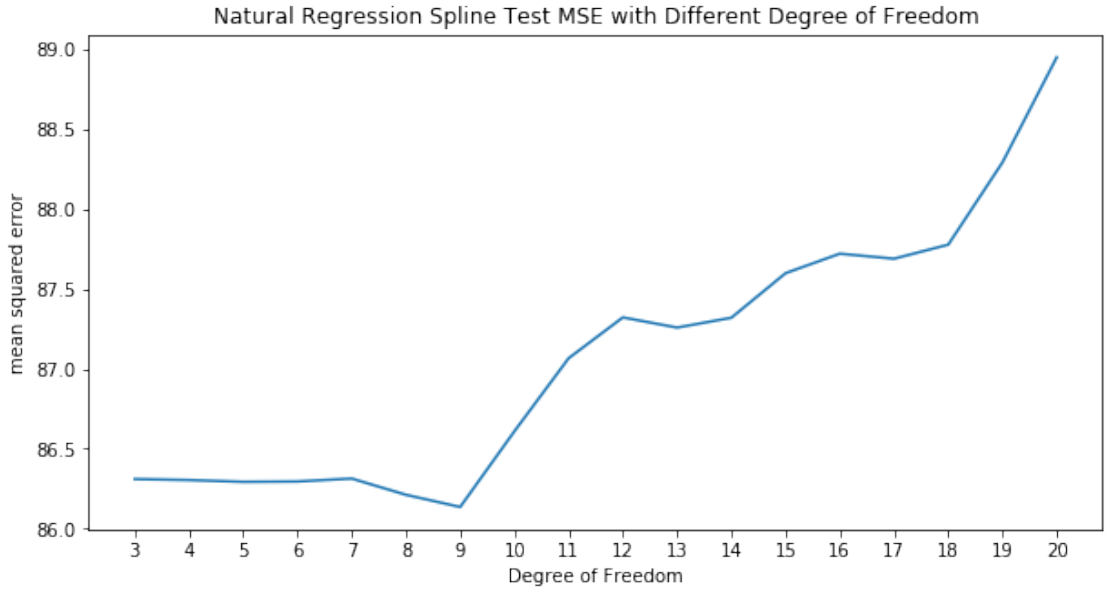
```
Best polynomial regression MSE: 85.89521226265285
Best step function regression MSE: 85.88881876728644
```

Using step function regression, we found out that the best bin number is 7 and either a higher or lower number would result in an increase of the test MSE. Compared to the result of the best polynomial regression model, it performs a little better in regard to the test MSE. Also, the regression result suggests a negative relationship between income06 and egalitarianism scale.

Q3 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 [189]:
```
# using python to conduct spline regression need to combine the spline
# basis in patsy and linear model in sklearn
from patsy import dmatrix
spline_result = []
all_income =list(gss_train['income06']) + list(gss_test['income06'])
x = np.unique(all_income).reshape(-1, 1)
neg_mse = sklearn.metrics.make_scorer(
sklearn.metrics.mean_squared_error, greater_is_better = False)
for d in range(3,21):
    # caculating the spline basis
    train_spline_features = dmatrix("cr(x, df={}) - 1".format(d),
    {"x": all_income})[:len(gss_train)]

    # building cross-validation regression model
    model = sklearn.linear_model.RidgeCV(
        alphas=[0.0], scoring=neg_mse, normalize=True, cv=10)
    model.fit(train_spline_features, gss_train['egalit_scale'])

    # prediction and testing
    test_spline_features = dmatrix("cr(x, df={}) - 1".format(d),
    {"x": all_income})[len(gss_train):]
    test_predict = model.predict(test_spline_features)
    test_mse = sklearn.metrics.mean_squared_error(
    gss_test['egalit_scale'], test_predict)

    spline_result.append((d, test_mse, model))
```

In [190]:
```
# plotting the mse variance with different cut points
plt.rcParams["figure.figsize"] = (10,5)
plt.plot([x[0] for x in spline_result],
         [x[1] for x in spline_result])
plt.title('Natural Regression Spline Test MSE with Different Degree of Freedom')
plt.xticks(range(3,21))
plt.xlabel('Degree of Freedom')
plt.ylabel('mean squared error')
plt.show()
```



In [191]:
```
# plotting the best fitting
plt.rcParams["figure.figsize"] = (10,5)
plt.scatter(gss_test.groupby(['income06']).mean()['egalit_scale'].index,
            gss_test.groupby(['income06']).mean()['egalit_scale'],
            label='data average' )

Best_Spline = sorted(spline_result, key= lambda x:x[1])[0]
Best_Spline_Model = Best_Spline[2]

x = np.unique(gss_test['income06']).reshape(-1, 1)
plt.plot(x, Best_Spline_Model.predict(
dmatrix("cr(x, df={}) - 1".format(Best_Spline[0]), {"x": x})),
label = 'degree of freedom = {} prediction'.format(Best_Spline[0]),
color = 'red')
plt.title('Best Spline Function Regression Data Fitting')
plt.xlabel('income level')
plt.ylabel('egalitarianism scale')
plt.legend()
plt.show()
```

0.1.2 Egalitarianism and everything

Q4 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 [3]: # data conversion
        train_object = gss_train.select_dtypes(['object']).columns
        gss_train[train_object] = gss_train[train_object].astype(
        'category').apply(lambda x: x.cat.codes)
        test_object = gss_test.select_dtypes(['object']).columns
        gss_test[test_object] = gss_test[test_object].astype(
        'category').apply(lambda x: x.cat.codes)

        gss_train_features = gss_train.drop(columns='egalit_scale')
        gss_test_features = gss_test.drop(columns='egalit_scale')
```

```
In [49]: # a. Linear regression

         mse_score = sklearn.metrics.make_scorer(
         sklearn.metrics.mean_squared_error)

         # Building the model
         OLS_model = LinearRegression(normalize = True).fit(
         gss_train_features, gss_train['egalit_scale'])
```

```
        # cross validation
        OLS_scores = sklearn.model_selection.cross_val_score(
        OLS_model, gss_test_features, gss_test['egalit_scale'],
        scoring = mse_score, cv=10)

        print("OLS test MSE validation scores:",
              *OLS_scores,sep="\n")
        print("OLS average test MSE:", np.mean(OLS_scores))
```

```
OLS test MSE validation scores:
65.062393047012
67.08170209735084
55.40169963021787
70.74395343147907
60.92441347275895
75.36099817451729
64.2174544799844
72.58770204349592
66.52506969263624
81.79101593458789
OLS average test MSE: 67.96964020040404
```

```
In [52]: # b. Elastic net regression

        from sklearn.linear_model import ElasticNetCV

        # Building the model and also tuning the parameters via cross-validation
        ElasticNet_model = ElasticNetCV(l1_ratio = np.linspace(0.1,1.0,10),
                           normalize = True, cv = 10, n_jobs = -1).fit(
                           gss_train_features, gss_train['egalit_scale'])

        # cross validation to estimate the test MSE
        ElasticNet_scores = sklearn.model_selection.cross_val_score(
        ElasticNet_model, gss_test_features, gss_test['egalit_scale'],
        scoring = mse_score, cv=10)

        print("ElasticNet test MSE validation scores:",
              *ElasticNet_scores, sep='\n')
        print("ElasticNet average test MSE:", np.mean(ElasticNet_scores))
```

```
ElasticNet test MSE validation scores:
65.58376867904326
59.09430487968439
61.6499859389375
69.82202003200635
49.93359162058504
70.16210481424982
```

```
66.9575281247693
68.30835834276124
60.11650350801416
83.17847648641725
ElasticNet average test MSE: 65.48066424264684
```

```
In [6]: # c. Principal component regression
        from sklearn.decomposition import PCA
        pcr_result = []
        # Building the model and tuning the parameter m
        for m in range(1,len(gss_train_features.columns)+1):
            pca = PCA(n_components = m)
            pca.fit(gss_train_features.append(gss_test_features))
            reduced_train = pca.transform(gss_train_features)
            reduced_test = pca.transform(gss_test_features)

            linear_model = LinearRegression(normalize = True).fit(
            reduced_train, gss_train['egalit_scale'])
            test_mse = sklearn.metrics.mean_squared_error(
            gss_test['egalit_scale'], linear_model.predict(reduced_test))

            pcr_result.append((m, linear_model, test_mse))
```

```
In [50]: best_pcr = sorted(pcr_result, key = lambda x:x[2])[0]

         # cross validation
         best_pca = PCA(n_components = best_pcr[0]).fit(
             gss_train_features.append(gss_test_features))
         PCR_scores = sklearn.model_selection.cross_val_score(
         best_pcr[1], best_pca.transform(gss_test_features),
         gss_test['egalit_scale'],scoring = mse_score, cv=10)

         print("PCR test MSE validation scores:",
               *PCR_scores, sep='\n')
         print("PCR average test MSE:", np.mean(PCR_scores))
```

```
PCR test MSE validation scores:
63.74222403481952
61.24914781823308
56.66233358517781
65.47772808231106
50.91423388324038
74.00896037405347
65.1496724523521
72.88499585666011
57.1445462957345
82.5985361565381
```

```
PCR average test MSE: 64.98323785391202
```

```
In [8]: # d. Partial least squares regression

        from sklearn.cross_decomposition import PLSRegression

        pls_result = []
        # Building the model and tuning the parameter m
        for m in range(1,len(gss_train_features.columns)+1):
            PLS_model = PLSRegression(n_components = m).fit(
            gss_train_features, gss_train['egalit_scale'])
            test_mse = sklearn.metrics.mean_squared_error(
            gss_test['egalit_scale'], PLS_model.predict(gss_test_features))
            pls_result.append((m, PLS_model, test_mse))
```

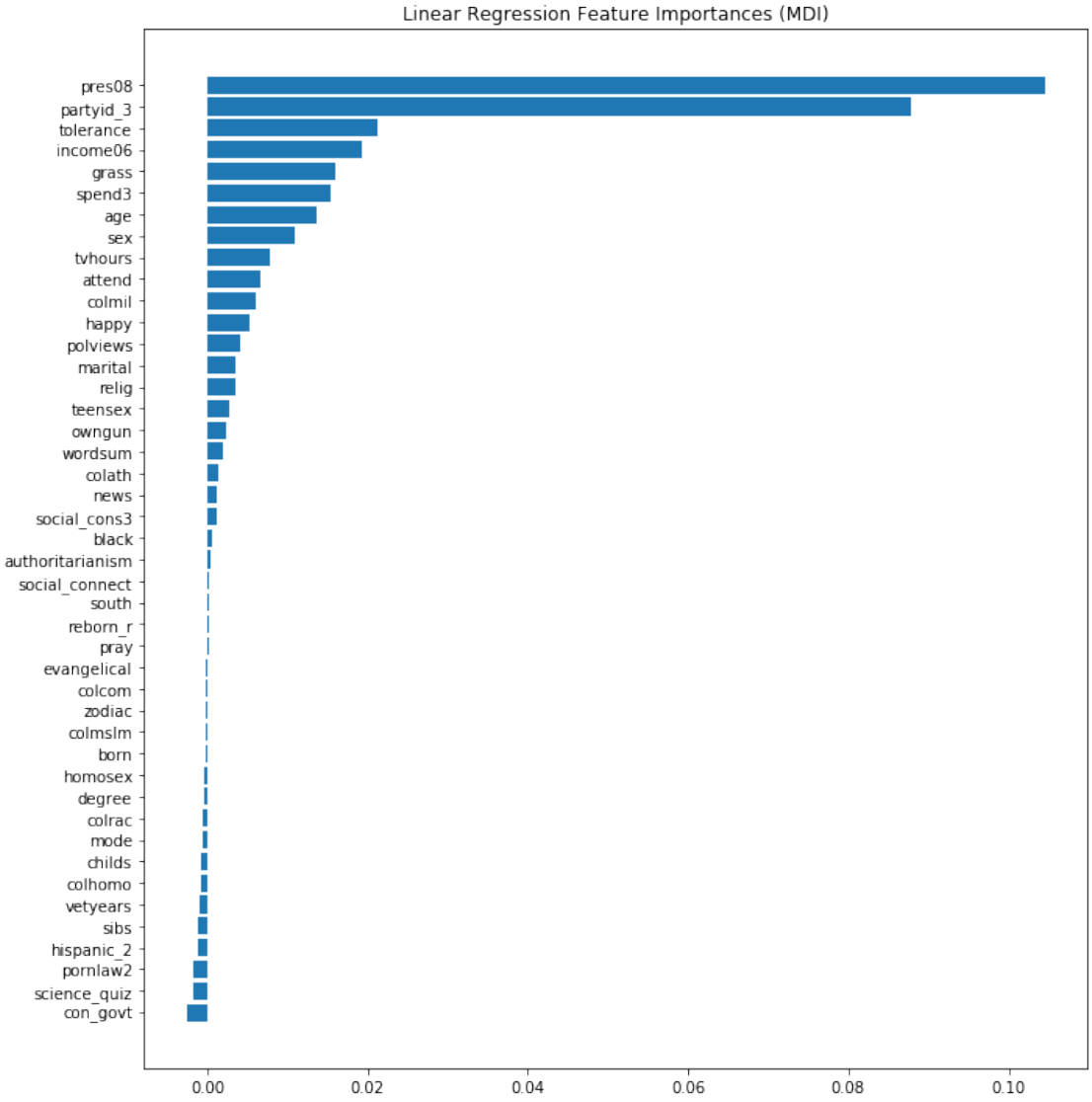
```
In [51]: best_pls = sorted(pls_result, key = lambda x:x[2])[0]

         # cross validation
         PLS_scores = sklearn.model_selection.cross_val_score(
         best_pls[1], gss_test_features, gss_test['egalit_scale'],
         scoring = mse_score, cv=10)

         print("PLS test MSE validation scores:",
               *PLS_scores, sep='\n')
         print("PLS average test MSE:", np.mean(PLS_scores))
```

```
PLS test MSE validation scores:
66.35420881669673
63.429668574890044
52.73664279202834
66.67516045213259
50.87539750217155
71.08106748569006
63.69929285352618
67.90936753718636
67.60497540022793
75.15438807592251
PLS average test MSE: 64.55201694904721
```

Q5 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.).

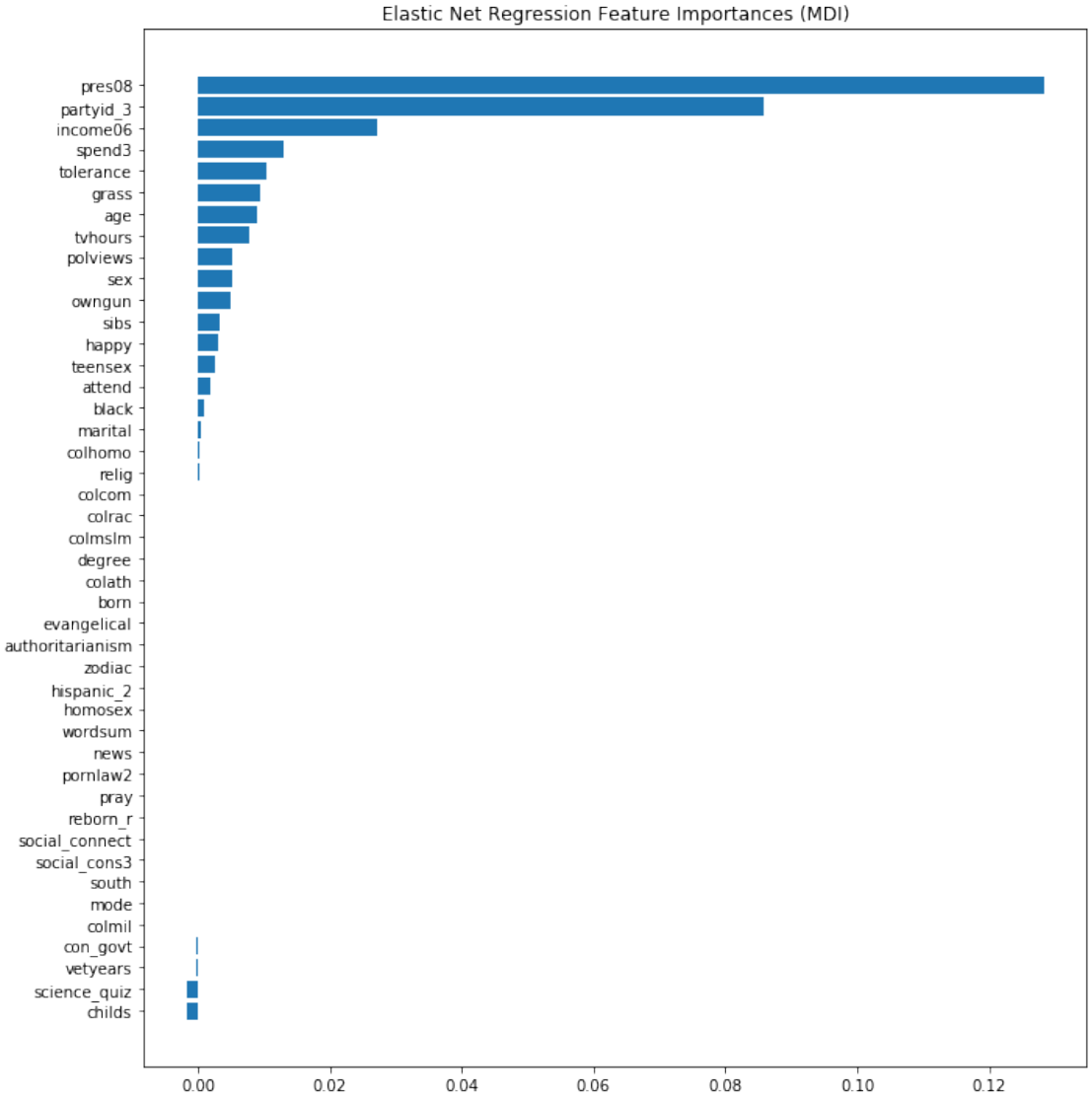
```
In [25]: from sklearn.inspection import permutation_importance
```

```
plt.rcParams["figure.figsize"] = (10,10)
# plotting the feature importance of the linear regression on the test set
OLS_importance = permutation_importance(
OLS_model,gss_test_features,gss_test['egalit_scale'])

sorted_idx = OLS_importance.importances_mean.argsort()
feature_names = gss_test_features.columns
feature_importances = OLS_importance.importances_mean

y_ticks = np.arange(0, len(feature_names))
fig, ax = plt.subplots()
ax.barh(y_ticks, feature_importances[sorted_idx])
ax.set_yticklabels(feature_names[sorted_idx])
ax.set_yticks(y_ticks)
ax.set_title("Linear Regression Feature Importances (MDI)")
fig.tight_layout()
plt.show()
```



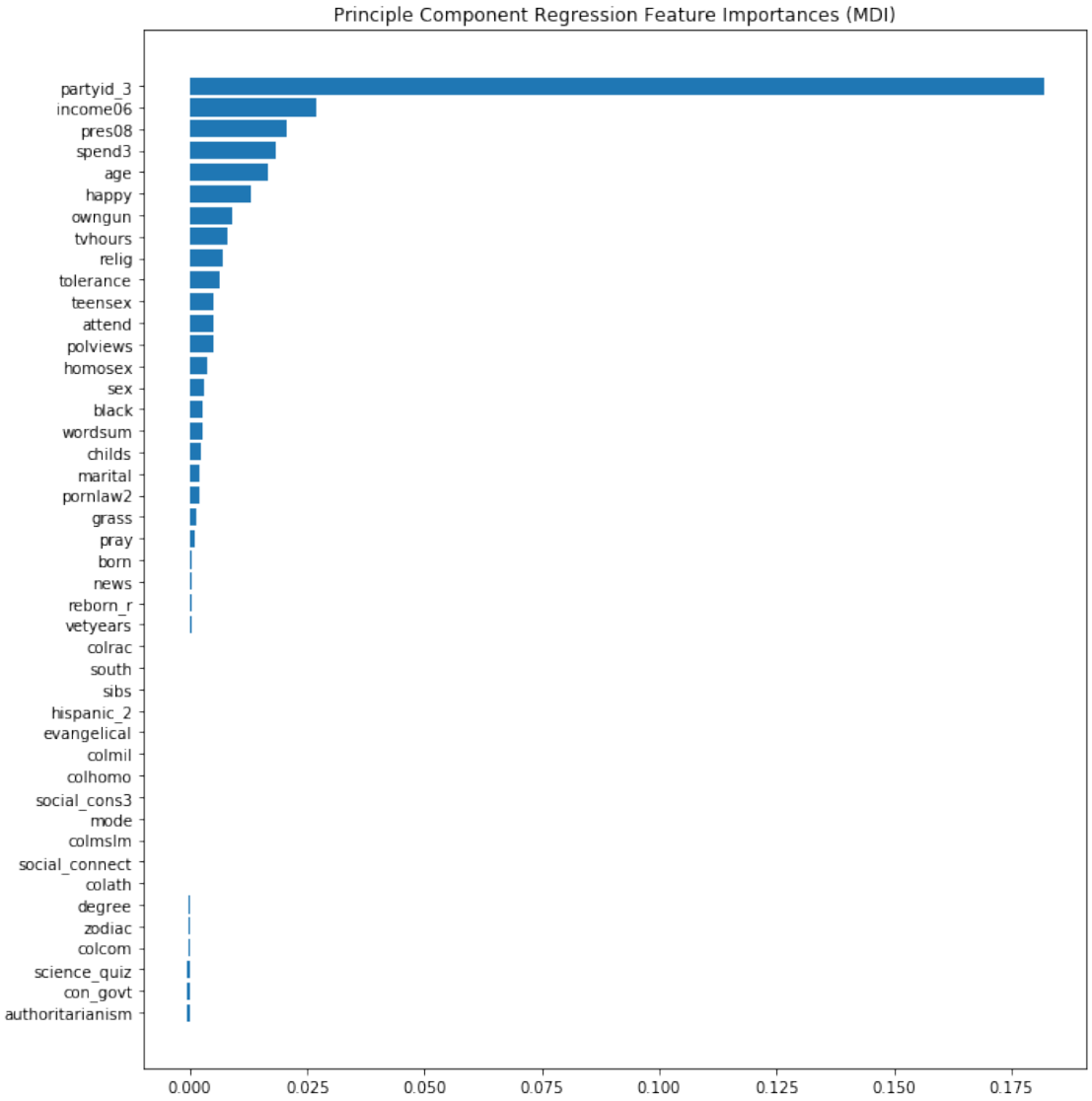
In [26]:
```
# plotting the feature importance of
# the ElasticNet regression on the test set
ElasticNet_importance = permutation_importance(
ElasticNet_model,gss_test_features,gss_test['egalit_scale'])

sorted_idx = ElasticNet_importance.importances_mean.argsort()
feature_names = gss_test_features.columns
feature_importances = ElasticNet_importance.importances_mean

y_ticks = np.arange(0, len(feature_names))
fig, ax = plt.subplots()
ax.barh(y_ticks, feature_importances[sorted_idx])
ax.set_yticklabels(feature_names[sorted_idx])
```

```
ax.set_yticks(y_ticks)
ax.set_title("Elastic Net Regression Feature Importances (MDI)")
fig.tight_layout()
plt.show()
```

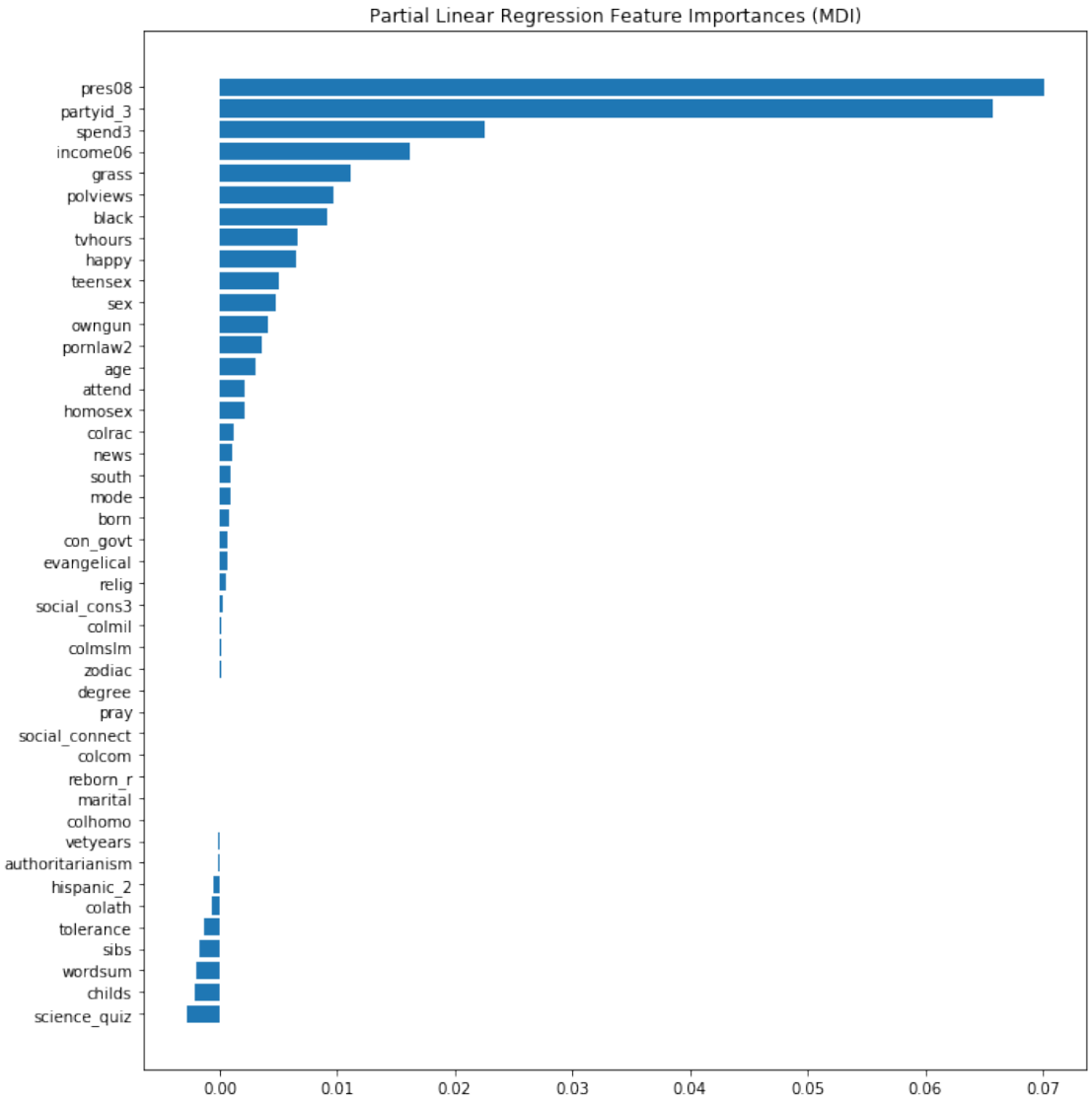


In [47]:
```
# plotting the feature importance of the
# Principal component regression on the test set
from sklearn.pipeline import make_pipeline

integrated_pcr = make_pipeline(
PCA(n_components = best_pcr[0]).fit(
gss_train_features.append(gss_test_features)),
best_pcr[1])
```

```
pcr_importance = permutation_importance(
integrated_pcr, gss_test_features, gss_test['egalit_scale'])

sorted_idx = pcr_importance.importances_mean.argsort()
feature_names = gss_test_features.columns
feature_importances = pcr_importance.importances_mean

y_ticks = np.arange(0, len(feature_names))
fig, ax = plt.subplots()
ax.barh(y_ticks, feature_importances[sorted_idx])
ax.set_yticklabels(feature_names[sorted_idx])
ax.set_yticks(y_ticks)
ax.set_title("Principle Component Regression Feature Importances (MDI)")
fig.tight_layout()
plt.show()
```



In [38]:
```
# plotting the feature importance of the PLS regression on the test set
pls_importance = permutation_importance(
best_pls[1],gss_test_features,gss_test['egalit_scale'])

sorted_idx = pls_importance.importances_mean.argsort()
feature_names = gss_test_features.columns
feature_importances = pls_importance.importances_mean

y_ticks = np.arange(0, len(feature_names))
fig, ax = plt.subplots()
ax.barh(y_ticks, feature_importances[sorted_idx])
ax.set_yticklabels(feature_names[sorted_idx])
ax.set_yticks(y_ticks)
ax.set_title("Partial Linear Regression Feature Importances (MDI)")
fig.tight_layout()
plt.show()
```



From all these four feature importance plots, we see that pres08 ("Vote obama or mccain"), partyid_3 ("Party id of person") and income06 ("Total family income") are always among the top ones. So it is reasonable to believe that these 3 attributes have strong correlations to respondents' egalitarianism scale. However, contrary to what we have found in previous questions, the relationship between the total family income and egalitarianism scale changes from negative to positive when all the attributes are applied to regressions. In this case I guess that change is caused by the interactions between income06 and other features, and these interactions might be stronger than the direct relationship between egalitarianism scale and income06. Also, when we take a look at those who take a relatively higher negative effect, we would find that science_quiz is the one that has been shared in all four regression models. For other attributes, especially those in the middle part, they are likely to pose little (or even zero in some cases) impact to the prediction.