```
The best value of k is k = 3. The RMSE indicated above

========== Question 3.(o) ==========
We try not to scale each feature to have unit variance but having range [0
, 1]That is, for each feature, we map value using function y = (x-a)/(b-a)
where a is the min value of the feature and b is the max value of the feat
ure
It does not helps much

========== Question 3.(p) ==========
The best model so far is kNN with k = 3, weighted neighbors and scaling
Done and submitted file submission.txt to gradescope.

========== Question 3.(q) ==========
The best naive classifier will assign all points to the most possible clas
s.
The answer is 1/k.

========== Question 3.(s) ==========
The SVM may not work well, since a lot of points mingle with each other.
In other words, they are not linearly separate.

========== Question 3.(t) ==========
SVM Classification
Accuracy: 0.75
Pipeline(memory=None,
     steps=[('svm', SVC(C=48.0, cache_size=200, class_weight=None, coef0=0
.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

The accuracy indicated above.

========== Question 3.(u) ==========
SVM Classification modified by adding PCA step and Scaling step
Accuracy: 0.810810810811
Pipeline(memory=None,
     steps=[('pca', PCA(copy=True, iterated_power='auto', n_components=7,
random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)), ('scale', StandardScaler(cop
y=True, with_mean=True, with_std=True)), ('svm', SVC(C=0.02, cache_size=20
0, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

The accuracy indicated above. It does improve.

========== Question 3.(v) ==========
SVM Classification with kernel of radial basis function
Accuracy: 0.689189189189
Pipeline(memory=None,
```

```
     steps=[('svm', SVC(C=98.0, cache_size=200, class_weight=None, coef0=0
.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False))])

The accuracy indicated above.

========== Question 3.(w) ==========
k Nearest Neighbors Classification
Accuracy: 0.763513513514
Pipeline(memory=None,
     steps=[('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, m
etric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])

The accuracy indicated above.

k Nearest Neighbors Classification with Scaling
Accuracy: 0.77027027027
Pipeline(memory=None,
     steps=[('scale', StandardScaler(copy=True, with_mean=True, with_std=T
rue)), ('knn', KNeighborsClassifier(algorithm='auto', leaf_size=30, metric
='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=4, p=2,
           weights='distance'))])

The accuracy indicated above.
Scling helps a little bit, increasing the accuracy from 0.7635 to 0.7703.

========== Question 3.(x) ==========
At 110 responses, the feature numbers for Berkeley are: [33,30,15,57,51,95
,44,55,36,57,22,54,31,47,36,52].
Predicted HDI: 0.461942596688

At 162 responses, the feature numbers for Berkeley are: [49,37,19,79,71,13
9,60,83,55,84,31,83,38,73,41,72].
Predicted HDI: 0.462173616823

At 229 responses, the feature numbers for Berkeley are: [68,46,26,116,98,1
98,83,115,78,118,39,116,58,99,64,89].
Predicted HDI: 0.461521885868

========== Question 3.(y) ==========
Regarding the sensor location problem, we can use kNN in the same way:Basi
cally, we are given the distances from m sensors, we can treat them as a v
ector of featuresFor kNN, we don't learn the model but we learn the bounda
ries (generative model vs discrimimative modelthat is, given a test point,
we determined its 'distance' from k training points, then we infer its loc
ationImplementation:1. Create kNN model knn = KNeighborsRegressor()2. Trai
n model knn.fit(X, y)3. Test model and caculate RMSE knn(X_test)This is th
e basic model, we expect to tune parameter k, attempt scaling, attempt wei
ghted neighbors, etc.
```

========== Question 3.(z) ==========
From this problems, I learned that data modelling is so painful and requir es a lot of patience.
Basically, we have to try many model, and we have to search for wide range s of different parametersbefore come up with an acceptable model.For the n ature of the problem, it looks like if our data spread out with low correl ations coefficient,the kNN method works better than ridge regression or la sso regression.Feedback for the problem author: The problem is very intere sting and useful.

```
"""
The world_values data set is available online at http://54.227.246.164/dataset/. In the data,
    residents of almost all countries were asked to rank their top 6 'priorities'. Specifically,
    they were asked "Which of these are most important for you and your family?"

This code and world-values.tex guides the student through the process of training several models
    to predict the HDI (Human Development Index) rating of a country from the responses of its
    citizens to the world values data. The new model they will try is k Nearest Neighbors (kNN).
    The students should also try to understand *why* the kNN works well.
"""

from math import sqrt
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsRegressor

from world_values_utils import import_world_values_data
from world_values_utils import hdi_classification
from world_values_utils import calculate_correlations
from world_values_utils import plot_pca
from world_values_utils import plot_pca_2

from world_values_pipelines import ridge_regression_pipeline
from world_values_pipelines import lasso_regression_pipeline
from world_values_pipelines import k_nearest_neighbors_regression_pipeline
```

```
from world_values_pipelines import k_nearest_neighbors_regression_scaled_pipeline
from world_values_pipelines import svm_classification_pipeline
from world_values_pipelines import svm_classification_pca_scale_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_pipeline
from world_values_pipelines import k_nearest_neighbors_classification_scale_pipeline

from world_values_parameters import regression_ridge_parameters
from world_values_parameters import regression_lasso_parameters
from world_values_parameters import regression_knn_parameters
from world_values_parameters import regression_knn_weighted_parameters
from world_values_parameters import classification_svm_parameters
from world_values_parameters import classification_svm_pca_scale_parameters
from world_values_parameters import classification_svm_rbf_parameters
from world_values_parameters import classification_knn_parameters



def main():
    print("========== Question 3.(a) ==========")
    print('Done filling out the "Berkeley F2017 Values Survey"')
    print()

    print("Predicting HDI from World Values Survey")
    print()

    # Import Data #
    print("Importing Training and Testing Data")
    values_train, hdi_train, values_test = import_world_values_data()

    # Center the HDI Values #
```

```
hdi_scaler = StandardScaler(with_std=False)
hdi_shifted_train = hdi_scaler.fit_transform(hdi_train)

# Classification Data #
hdi_class_train = hdi_train['2015'].apply(hdi_classification)

# Data Information #
print('Training Data Count:', values_train.shape[0])
print('Test Data Count:', values_test.shape[0])
print()

# Calculate Correlations #
print("========== Question 3.(b)(c) ==========")
correlations = calculate_correlations(values_train, hdi_train)

# PCA #
print("========== Question 3.(d) ==========")
plot_pca(values_train, hdi_train, hdi_class_train)
print()

# Regression Grid Searches #
regression_grid_searches(training_features=values_train,
                         training_labels=hdi_shifted_train)

print("========== Question 3.(o) ==========")
print("We try not to scale each feature to have unit variance but having range [0, 1]" +
   "That is, for each feature, we map value using function y = (x-a)/(b-a)" +
   "where a is the min value of the feature and b is the max value of the feature")
mapping = lambda x, a, b: (x-a)/(b-a)
```

```
mapped_values_train = scaler.fit_transform(values_train)
mapped_values_test = scaler.transform(values_test)
knn_map = KNeighborsRegressor(n_neighbors=3, weights="distance")
knn_map.fit(mapped_values_train, hdi_train["2015"])
knn.predict(scaled_values_test)
print("It does not helps much")
print()

print("========== Question 3.(p) ==========")
print("The best model so far is kNN with k = 3, weighted neighbors and scaling")
scaler = StandardScaler()
scaled_values_train = scaler.fit_transform(values_train)
scaled_values_test = scaler.transform(values_test)
knn = KNeighborsRegressor(n_neighbors=3, weights="distance")
knn.fit(scaled_values_train, hdi_train["2015"])
hdi_test = knn.predict(scaled_values_test)
with open("submission.txt", "w") as f:
    for i in hdi_test:
        f.write(str(i)[:6] + "\n")
print("Done and submitted file submission.txt to gradescope.")
print()

print("========== Question 3.(q) ==========")
print("The best naive classifier will assign all points to the most possible class.\n" +
    "The answer is 1/k.")
print()

# PCA for Classification#
# print("========== Question 3.(r) ==========")
```

```
    # plot_pca_2(values_train, hdi_train, hdi_class_train)
    # print()

    print("========== Question 3.(s) ==========")
    print("The SVM may not work well, since a lot of points mingle with each other.\n" +
        "In other words, they are not linearly separate.")
    print()

    # Classification Grid Searches #
    classification_grid_searches(training_features=values_train,
                                 training_classes=hdi_class_train)

    print("========== Question 3.(x) ==========")
    print("At 110 responses, the feature numbers for Berkeley are:
[33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52].")
    berkeley_110 = [33,30,15,57,51,95,44,55,36,57,22,54,31,47,36,52]
    berkeley_110 = [i / sum(berkeley_110) for i in berkeley_110]
    scaled_berkeley_110 = scaler.transform([berkeley_110])
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_110)[0]))
    print()
    print("At 162 responses, the feature numbers for Berkeley are:
[49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72].")
    berkeley_162 = [49,37,19,79,71,139,60,83,55,84,31,83,38,73,41,72]
    berkeley_162 = [i / sum(berkeley_162) for i in berkeley_162]
    scaled_berkeley_162 = scaler.transform([berkeley_162])
    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_162)[0]))
    print()
    print("At 229 responses, the feature numbers for Berkeley are:
[68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89].")
    berkeley_229 = [68,46,26,116,98,198,83,115,78,118,39,116,58,99,64,89]
```

```
    berkeley_229 = [i / sum(berkeley_229) for i in berkeley_229]

    scaled_berkeley_229 = scaler.transform([berkeley_229])

    print("Predicted HDI: " + str(knn.predict(scaled_berkeley_229)[0]))

    print()


    print("========== Question 3.(y) ==========")

    print("Regarding the sensor location problem, we can use kNN in the same way:" +

        "Basically, we are given the distances from m sensors, we can treat them as a vector of features" +

        "For kNN, we don't learn the model but we learn the boundaries (generative model vs
discrimimative model" +

        "that is, given a test point, we determined its 'distance' from k training points, then we infer its
location" +

        "Implementation:" +

        "1. Create kNN model knn = KNeighborsRegressor()" +

        "2. Train model knn.fit(X, y)" +

        "3. Test model and caculate RMSE knn(X_test)" +

        "This is the basic model, we expect to tune parameter k, attempt scaling, attempt weighted
neighbors, etc.")

    print()


    print("========== Question 3.(z) ==========")

    print("From this problems, I learned that data modelling is so painful and requires a lot of patience.\n"
+

        "Basically, we have to try many model, and we have to search for wide ranges of different
parameters" +

        "before come up with an acceptable model." +

        "For the nature of the problem, it looks like if our data spread out with low correlations
coefficient,"+

        "the kNN method works better than ridge regression or lasso regression." +

        "Feedback for the problem author: The problem is very interesting and useful.")

    print()
```

```
def find_neighbors(training_features):
    distance_map = {}
    usa = np.array(training_features.iloc[45])
    for i in range(training_features.shape[0]):
        country = np.array(training_features.iloc[i])
        distance_map[i] = np.mean( (country - usa) ** 2 ) ** 0.5

    usa_neighbors = []
    for _ in range(8):
        index = min(distance_map, key=distance_map.get)
        distance_map.pop(index)
        usa_neighbors.append(index)
    print("Country indices: " + str(usa_neighbors[1:]))



def _rmse_grid_search(training_features, training_labels, pipeline, parameters, technique):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        pipeline: regression model specific pipeline
        parameters: regression model specific parameters
        technique: regression model's name

    Output:
        Prints best RMSE and best estimator
        Prints feature weights for Ridge and Lasso Regression
```

```
    Plots RMSE vs k for k Nearest Neighbors Regression
    """
    grid = GridSearchCV(estimator=pipeline,
                        param_grid=parameters,
                        scoring='neg_mean_squared_error')
    grid.fit(training_features,
             training_labels)
    print("RMSE:", sqrt(-grid.best_score_))
    print(grid.best_estimator_)

    # Check Ridge or Lasso Regression
    if hasattr(grid.best_estimator_.named_steps[technique], 'coef_'):
        print(grid.best_estimator_.named_steps[technique].coef_)
    else:
        # Plot RMSE vs k for k Nearest Neighbors Regression
        plt.plot(grid.cv_results_['param_knn__n_neighbors'],
                 (-grid.cv_results_['mean_test_score'])**0.5)
        plt.xlabel('k')
        plt.ylabel('RMSE')
        plt.title('RMSE versus k in kNN')
        plt.show()

    print()



def regression_grid_searches(training_features, training_labels):
    """
    Input:
        training_features: world_values responses on the training set
```

```
    training_labels: HDI (human development index) on the training set

    Output:
        Prints best RMSE, best estimator, feature weights for Ridge and Lasso Regression
        Prints best RMSE, best estimator, and plots RMSE vs k for k Nearest Neighbors Regression
    """

    print("========== Question 3.(e) ==========")
    print("Ridge Regression")
    _rmse_grid_search(training_features, training_labels,
            ridge_regression_pipeline, regression_ridge_parameters, 'ridge')
    print("I changed the range of hyper-parameter ridge__alpha to obtain the finer result. That is\n" +
        "ridge__alpha has np.arange(0.001, 1.0, 0.001) instead of np.arange(0.01, 1.0, 0.01) given\n" +
        "The best RMSE indicated above.")
    print()

    print("========== Question 3.(f) ==========")
    print("Lasso Regression")
    _rmse_grid_search(training_features, training_labels,
            lasso_regression_pipeline, regression_lasso_parameters, 'lasso')
    print("I changed the range of hyper-parameter lasso__alpha to obtain the finer result. That is\n" +
        "lasso__alpha has np.arange(0.00001, 0.01, 0.00001) instead of np.arange(0.0001, 0.01, 0.0001)
given\n" +
        "The best RMSE indicated above.")
    print()

    print("========== Question 3.(g) ==========")
    print("The Lasso Regression does give more 0 weights.\n" +
        "That indicates some features do not really matter in this method.")
```

```
    print()


    print("========== Question 3.(h) ==========")
    print("To deal with continuous outputs, we can weight the neigbors instead of treating them uniformly.\n" +
          "Say, each neighbor is weighted by its inverse distance, and we use the average of k-neareast-neighbor\n" +
          "weights to predict the output.")
    print()


    print("========== Question 3.(i) ==========")
    print("The 7 nearest neighbors of the USA:")
    find_neighbors(training_features)
    print("Countries: Ireland, United Kingdom, Belgium, Finland, Malta, Austria, France")
    print()


    print("========== Question 3.(j) ==========")
    print("k Nearest Neighbors Regression")
    _rmse_grid_search(training_features, training_labels,
                      k_nearest_neighbors_regression_pipeline,
                      regression_knn_parameters, 'knn')
    print("The best value of k is k = 12. The RMSE indicated above")
    print()


    print("========== Question 3.(k) ==========")
    print("When we increase k, the model goes from overfitting to fitting then underfitting. That is increasing k leads\n" +
          "to the increase of bias and decrease of variance because the model becomes less flexible to accommodate ambiguous\n" +
```

```
      "points. Since more points are taken into account when considering a test point, the model works more consistent\n" +
      "(with lower variance), but less accurate (with higher bias) if the nature of training data is the mingling of data points.")
   print()

   print("========== Question 3.(l) ==========")
   print("k Nearest Neighbors Regression with weighted neighbor distances")
   _rmse_grid_search(training_features, training_labels,
            k_nearest_neighbors_regression_pipeline,
            regression_knn_weighted_parameters, 'knn')
   print("The best value of k is k = 14. The RMSE indicated above")
   print()

   print("========== Question 3.(m) ==========")
   scaler = StandardScaler()
   scaled_training_features = scaler.fit_transform(training_features)
   scaled_training_features = pd.DataFrame(scaled_training_features)
   find_neighbors(scaled_training_features)
   print("Countries: Ireland, United Kingdom, Finland, Belgium, Malta, France, Austria")
   print()
   print("Compared to (i), the neighbors just change order a little bit.")
   print()

   print("========== Question 3.(n) ==========")
   print("k Nearest Neighbors Regression with weighted neighbor distances and scaling the features")
   _rmse_grid_search(training_features, training_labels,
            k_nearest_neighbors_regression_scaled_pipeline,
            regression_knn_weighted_parameters, 'knn')
```

```
    print("The best value of k is k = 3. The RMSE indicated above")
    print()

    print("========== Question 3.(o) ==========")
    print("The best model so far is kNN with k = 3, weighted neighbors and scaling")
    scaler = StandardScaler()
    scaled_values_train = scaler.fit_transform(values_train)
    scaled_values_test = scaler.transform(values_test)
    knn = KNeighborsRegressor(n_neighbors=3, weights="distance")
    knn.fit(scaled_values_train, hdi_train["2015"])
    hdi_test = knn.predict(scaled_values_test)
    with open("submission.txt", "w") as f:
        for i in hdi_test:
            f.write(str(i)[:6] + "\n")
    print("Done and submitted file submission.txt to gradescope.")
    print()



def _accuracy_grid_search(training_features, training_classes, pipeline, parameters):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set
        pipeline: classification model specific pipeline
        parameters: classification model specific parameters

    Output:
        Prints best accuracy and best estimator of classification model
    """
```

```
    grid = GridSearchCV(estimator=pipeline,
                        param_grid=parameters,
                        scoring='accuracy')
    grid.fit(training_features, training_classes)
    print("Accuracy:", grid.best_score_)
    print(grid.best_estimator_)
    print()



def classification_grid_searches(training_features, training_classes):
    """
    Input:
        training_features: world_values responses on the training set
        training_labels: HDI (human development index) on the training set

    Output:
        Prints best accuracy and best estimator for SVM and k Nearest Neighbors Classification
    """
    print("========== Question 3.(t) ==========")
    print("SVM Classification")
    _accuracy_grid_search(training_features, training_classes,
                        svm_classification_pipeline,
                        classification_svm_parameters)
    print("The accuracy indicated above.")
    print()

    print("========== Question 3.(u) ==========")
    print("SVM Classification modified by adding PCA step and Scaling step")
    _accuracy_grid_search(training_features, training_classes,
```

```
            svm_classification_pca_scale_pipeline,
            classification_svm_pca_scale_parameters)
print("The accuracy indicated above. It does improve.")
print()

print("========== Question 3.(v) ==========")
print("SVM Classification with kernel of radial basis function")
_accuracy_grid_search(training_features, training_classes,
            svm_classification_pipeline,
            classification_svm_rbf_parameters)
print("The accuracy indicated above.")
print()

print("========== Question 3.(w) ==========")
print("k Nearest Neighbors Classification")
_accuracy_grid_search(training_features, training_classes,
            k_nearest_neighbors_classification_pipeline,
            classification_knn_parameters)
print("The accuracy indicated above.")
print()

print("k Nearest Neighbors Classification with Scaling")
_accuracy_grid_search(training_features, training_classes,
            k_nearest_neighbors_classification_scale_pipeline,
            classification_knn_parameters)
print("The accuracy indicated above.")
print("Scling helps a little bit, increasing the accuracy from 0.7635 to 0.7703.")
print()
```

```
if __name__ == '__main__':
    main()
```