In [432]:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm
from sklearn.preprocessing import StandardScaler
from scipy import stats
import warnings
import missingno as mno
from sklearn.pipeline import Pipeline


#warnings.filterwarnings('ignore')
%matplotlib inline
```

In [218]:

```
df = pd.read_csv('LifeExpectancyData.csv')
df.head(10)
```

Out[218]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Me
0	Afghanistan	2015	Developing	65.0	263.0	62	0.01	71.279624	65.0	
1	Afghanistan	2014	Developing	59.9	271.0	64	0.01	73.523582	62.0	
2	Afghanistan	2013	Developing	59.9	268.0	66	0.01	73.219243	64.0	
3	Afghanistan	2012	Developing	59.5	272.0	69	0.01	78.184215	67.0	
4	Afghanistan	2011	Developing	59.2	275.0	71	0.01	7.097109	68.0	
5	Afghanistan	2010	Developing	58.8	279.0	74	0.01	79.679367	66.0	
6	Afghanistan	2009	Developing	58.6	281.0	77	0.01	56.762217	63.0	
7	Afghanistan	2008	Developing	58.1	287.0	80	0.03	25.873925	64.0	
8	Afghanistan	2007	Developing	57.5	295.0	82	0.02	10.910156	63.0	
9	Afghanistan	2006	Developing	57.3	295.0	84	0.03	17.171518	64.0	

10 rows × 22 columns

In [698]:

```
type(df['Country'])
```

Out[698]:

```
pandas.core.series.Series
```

In [219]:
```
df.sample(10)
```

Out[219]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	M
2104	Republic of Moldova	2014	Developing	71.8	162.0	1	9.99	0.000000	92.0	
1188	India	2013	Developing	67.6	187.0	1000	3.11	67.672304	7.0	
1685	Mexico	2013	Developing	76.6	12.0	32	5.23	150.408875	82.0	
136	Austria	2007	Developed	81.0	8.0	0	12.50	7453.864400	85.0	
1794	Myanmar	2001	Developing	62.5	239.0	72	0.38	1.917164	NaN	
2760	United Arab Emirates	2001	Developing	74.5	14.0	1	1.67	243.753913	92.0	
973	Gambia	2004	Developing	57.3	296.0	3	2.51	0.000000	95.0	
388	Bulgaria	2011	Developed	73.7	144.0	1	10.67	875.149519	96.0	
1646	Malta	2003	Developed	78.5	71.0	0	6.70	1678.392773	89.0	
260	Belize	2011	Developing	69.4	188.0	0	6.64	605.628689	95.0	

10 rows × 22 columns

In [220]:
```
print("Names of Columns : ")
print()
for x in df.columns :

    print("{}".format(x))

```

```
Names of Columns :

Country
Year
Status
Life expectancy
Adult Mortality
infant deaths
Alcohol
percentage expenditure
Hepatitis B
Measles
 BMI
under-five deaths
Polio
Total expenditure
Diphtheria
 HIV/AIDS
GDP
Population
 thinness  1-19 years
 thinness 5-9 years
Income composition of resources
Schooling
```

In [221]:
```
print("Number of ROWS in the dataset    : {}  ".format(df.shape[0]))
print("Number of COLUMNS in the dataset : {}  ".format(df.shape[1]))

```

```
Number of ROWS in the dataset    : 2938
Number of COLUMNS in the dataset : 22
```

In [222]:
```
df.describe()
```

Out[222]:

	Year	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis E
count	2938.000000	2928.000000	2928.000000	2938.000000	2744.000000	2938.000000	2385.00000(
mean	2007.518720	69.224932	164.796448	30.303948	4.602861	738.251295	80.94046
std	4.613841	9.523867	124.292079	117.926501	4.052413	1987.914858	25.07001(
min	2000.000000	36.300000	1.000000	0.000000	0.010000	0.000000	1.00000(
25%	2004.000000	63.100000	74.000000	0.000000	0.877500	4.685343	77.00000(
50%	2008.000000	72.100000	144.000000	3.000000	3.755000	64.912906	92.00000(
75%	2012.000000	75.700000	228.000000	22.000000	7.702500	441.534144	97.00000(
max	2015.000000	89.000000	723.000000	1800.000000	17.870000	19479.911610	99.00000(

In [223]:
```
df.info()

```

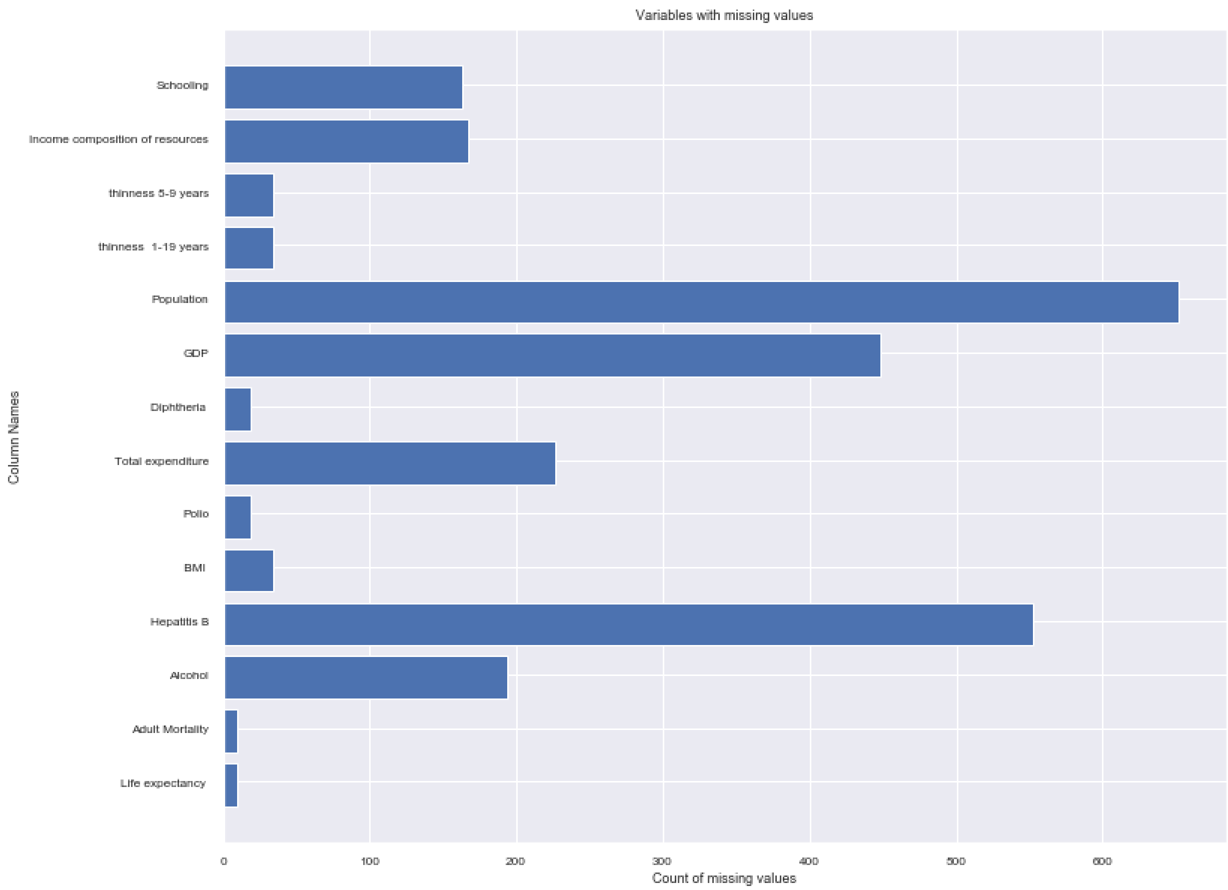
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2938 entries, 0 to 2937
Data columns (total 22 columns):
Country                            2938 non-null object
Year                               2938 non-null int64
Status                             2938 non-null object
Life expectancy                    2928 non-null float64
Adult Mortality                    2928 non-null float64
infant deaths                      2938 non-null int64
Alcohol                            2744 non-null float64
percentage expenditure             2938 non-null float64
Hepatitis B                        2385 non-null float64
Measles                            2938 non-null int64
 BMI                               2904 non-null float64
under-five deaths                  2938 non-null int64
Polio                              2919 non-null float64
Total expenditure                  2712 non-null float64
Diphtheria                         2919 non-null float64
 HIV/AIDS                          2938 non-null float64
GDP                                2490 non-null float64
Population                         2286 non-null float64
 thinness  1-19 years              2904 non-null float64
 thinness 5-9 years                2904 non-null float64
Income composition of resources    2771 non-null float64
Schooling                          2775 non-null float64
dtypes: float64(16), int64(4), object(2)
memory usage: 505.0+ KB
```

In [224]:
```
df.dtypes
```

Out[224]:
```
Country                            object
Year                                int64
Status                             object
Life expectancy                   float64
Adult Mortality                   float64
infant deaths                       int64
Alcohol                           float64
percentage expenditure            float64
Hepatitis B                       float64
Measles                             int64
 BMI                              float64
under-five deaths                   int64
Polio                             float64
Total expenditure                 float64
Diphtheria                        float64
 HIV/AIDS                         float64
GDP                               float64
Population                        float64
 thinness  1-19 years             float64
 thinness 5-9 years               float64
Income composition of resources   float64
Schooling                         float64
dtype: object
```

In [167]:

```
null_columns=df.columns[df.isnull().any()]
df[null_columns].isnull().sum()

labels = []
values = []
for col in null_columns:
    labels.append(col)
    values.append(df[col].isnull().sum())
ind = np.arange(len(labels))
width = 0.1
fig, ax = plt.subplots(figsize=(12,10))
rects = ax.barh(ind, np.array(values))
ax.set_yticks(ind+((width)/2.))
ax.set_yticklabels(labels, rotation='horizontal')
ax.set_xlabel("Count of missing values")
ax.set_ylabel("Column Names")
ax.set_title("Variables with missing values");
```

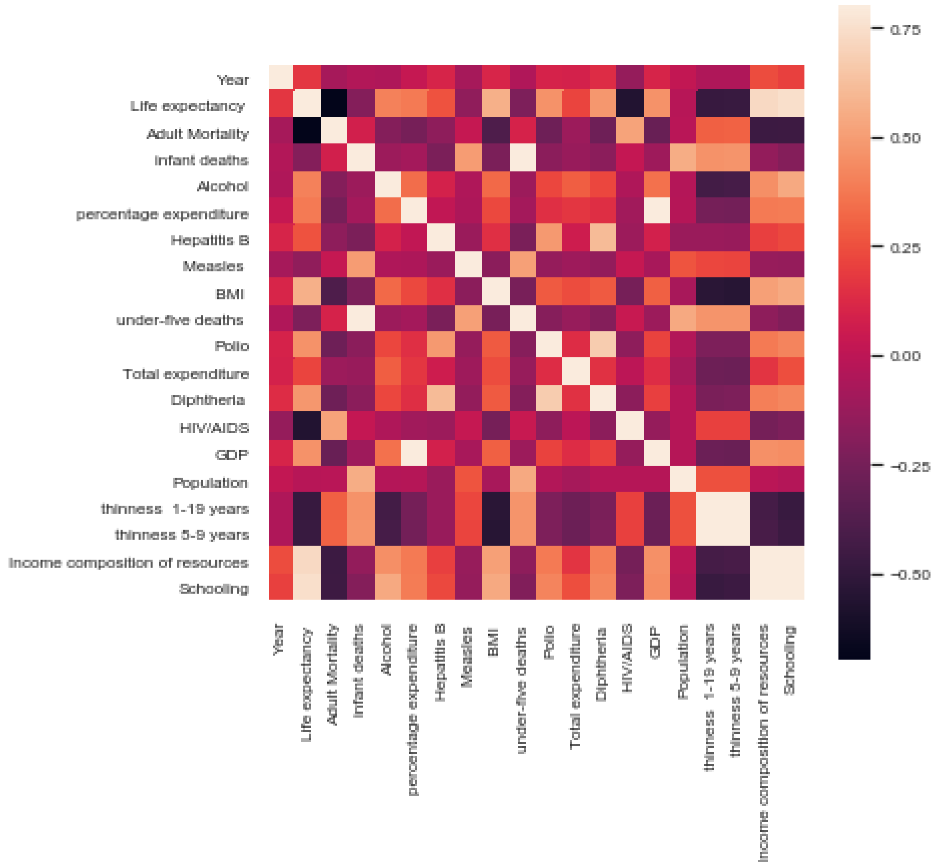


In [168]:
```
df.isna().sum().sort_values(ascending=False) / len(df) * 100

```

Out[168]:
```
Population                         22.191967
Hepatitis B                        18.822328
GDP                                15.248468
Total expenditure                   7.692308
Alcohol                             6.603131
Income composition of resources     5.684139
Schooling                           5.547992
 BMI                                1.157250
 thinness  1-19 years               1.157250
 thinness 5-9 years                 1.157250
Diphtheria                          0.646698
Polio                               0.646698
Adult Mortality                     0.340368
Life expectancy                     0.340368
under-five deaths                   0.000000
 HIV/AIDS                           0.000000
Measles                             0.000000
percentage expenditure              0.000000
infant deaths                       0.000000
Status                              0.000000
Year                                0.000000
Country                             0.000000
dtype: float64
```

In [169]:

```
corrmat = df.corr()
f, ax = plt.subplots(figsize=(6, 6))
sns.heatmap(corrmat, vmax=.8, square=True);
sns.set(font_scale=0.75)
```



In [170]:
```
correlations=df.corr()
attrs = correlations.iloc[:-1,:-1] # all except target

threshold = 0.5
important_corrs = (attrs[abs(attrs) > threshold][attrs != 1.0]) \
    .unstack().dropna().to_dict()

unique_important_corrs = pd.DataFrame(
    list(set([(tuple(sorted(key)), important_corrs[key]) \
    for key in important_corrs])),
        columns=['Attribute Pair', 'Correlation'])

    # sorted by absolute value
unique_important_corrs = unique_important_corrs.ix[
    abs(unique_important_corrs['Correlation']).argsort()[::-1]]

unique_important_corrs
```

Out[170]:

	Attribute Pair	Correlation
5	(infant deaths, under-five deaths)	0.996629
14	(thinness 1-19 years, thinness 5-9 years)	0.939102
15	(GDP, percentage expenditure)	0.899373
1	(Income composition of resources, Life expecta...	0.724776
10	(Adult Mortality, Life expectancy)	-0.696359
12	(Diphtheria , Polio)	0.673553
0	(Diphtheria , Hepatitis B)	0.611495
11	(BMI , Life expectancy)	0.567694
6	(Population, infant deaths)	0.556801
9	(HIV/AIDS, Life expectancy)	-0.556556
13	(Population, under-five deaths)	0.544423
4	(BMI , thinness 5-9 years)	-0.538911
2	(BMI , thinness 1-19 years)	-0.532025
16	(HIV/AIDS, Adult Mortality)	0.523821
8	(BMI , Income composition of resources)	0.508774
7	(Measles , under-five deaths)	0.507809
3	(Measles , infant deaths)	0.501128

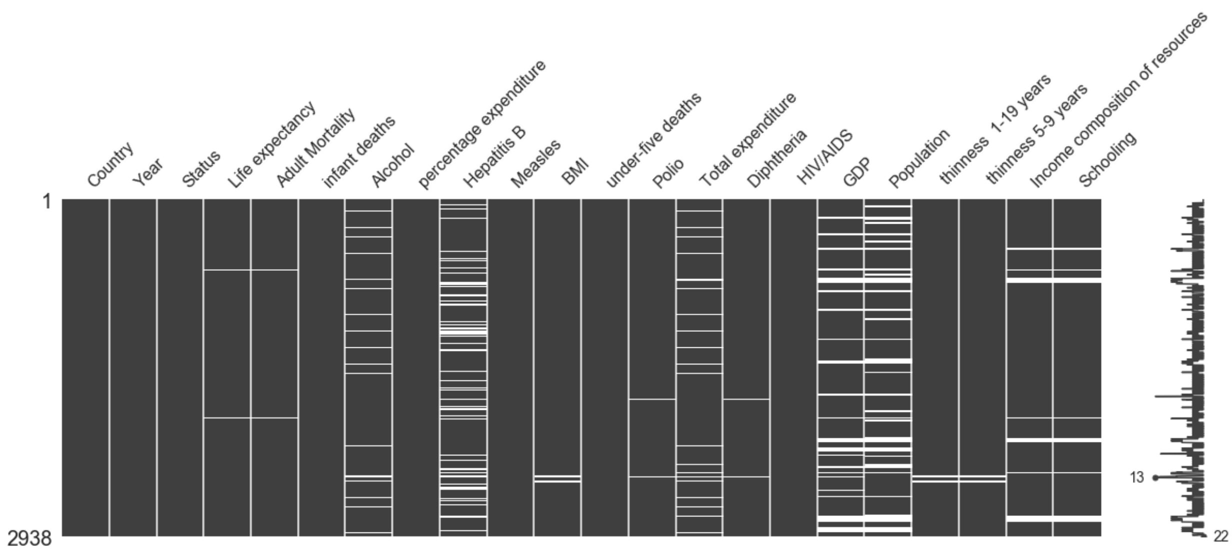
In [171]:
```
#num_feat
```

In [172]:
```
num_feat
```

Out[172]:
```
Index(['Year', 'Life expectancy ', 'Adult Mortality', 'infant deaths',
       'Alcohol', 'percentage expenditure', 'Hepatitis B', 'Measles ', '
BMI ',
       'under-five deaths ', 'Polio', 'Total expenditure', 'Diphtheria ',
       ' HIV/AIDS', 'GDP', 'Population', ' thinness  1-19 years',
       ' thinness 5-9 years', 'Income composition of resources', 'Schooli
ng'],
      dtype='object')
```

In [173]:
```
mno.matrix(df, figsize = (20, 6))
```

Out[173]: <matplotlib.axes._subplots.AxesSubplot at 0x1a295eae10>

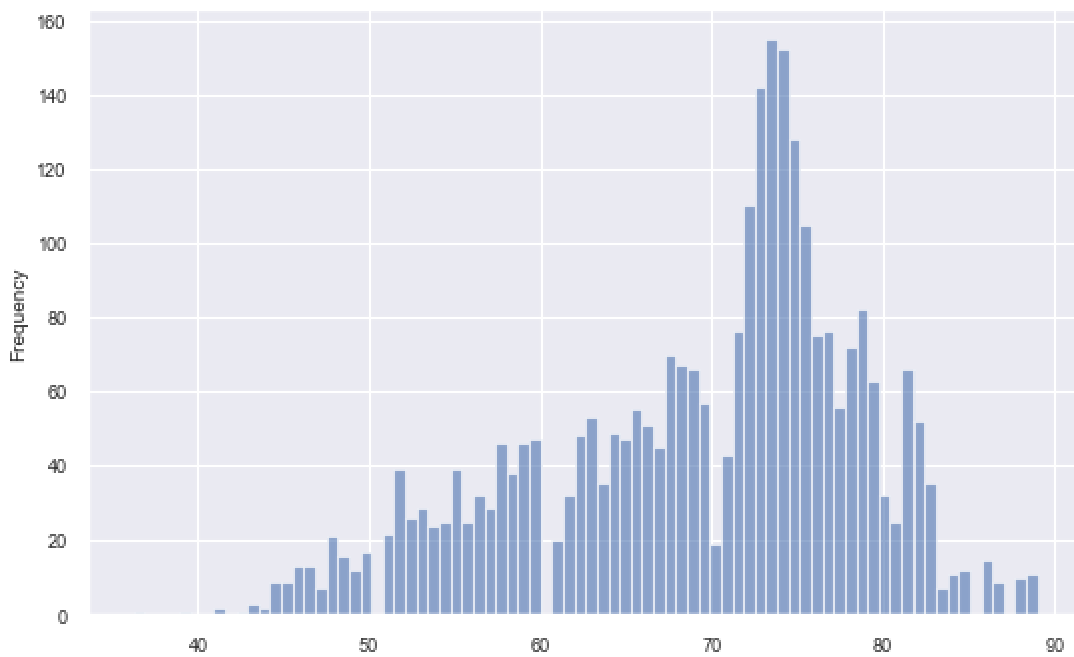


In [175]:
```
#np.corrcoef?
#(df['Year'].values, df['Life expectancy '].values)[0,0]
```

In [ ]:
```

```

In [176]:
```
plt.figure(num=None, figsize=(8, 5), dpi=80, facecolor='w', edgecolor='



```

Out[176]: <matplotlib.axes._subplots.AxesSubplot at 0x1a29e5ec50>



In [177]:
```
#df.dtypes

```

In [231]:
```
from sklearn.preprocessing import LabelEncoder

class MultiColumnLabelEncoder:
    def __init__(self,columns = None):
        self.columns = columns # array of column names to encode

    def fit(self,X,y=None):
        return self # not relevant here

    def transform(self,X):
        '''
        Transforms columns of X specified in self.columns using
        LabelEncoder(). If no columns specified, transforms all
        columns in X.
        '''
        output = X.copy()
        if self.columns is not None:
            for col in self.columns:
                output[col] = LabelEncoder().fit_transform(output[col])
        else:
            for colname,col in output.iteritems():
                output[colname] = LabelEncoder().fit_transform(col)
        return output

    def fit_transform(self,X,y=None):
        return self.fit(X,y).transform(X)
```

In [232]:
```
df = MultiColumnLabelEncoder(columns = ['Country','Status']).fit_transf

```

In [233]:
```
df.dtypes
```

Out[233]:
```
Country                            int64
Year                               int64
Status                             int64
Life expectancy                  float64
Adult Mortality                  float64
infant deaths                      int64
Alcohol                          float64
percentage expenditure           float64
Hepatitis B                      float64
Measles                            int64
 BMI                             float64
under-five deaths                  int64
Polio                            float64
Total expenditure                float64
Diphtheria                       float64
 HIV/AIDS                        float64
GDP                              float64
Population                       float64
 thinness  1-19 years            float64
 thinness 5-9 years              float64
Income composition of resources  float64
Schooling                        float64
dtype: object
```

In [322]:
```
class MultiColumnSimpleImputer:

    def __init__(self,columns = None):
        self.columns = columns # array of column names to encode

    def fit(self,X,y=None):
        return self # not relevant here

    def transform(self,X):
        output = X.copy()
        if (self.columns is not None):
            for col in self.columns:
                if ((output[col].dtypes == 'float64') or (output[col].c
                    output[col] = SimpleImputer(missing_values=np.nan,
                else:
                    output[col] = SimpleImputer(missing_values=np.nan,
        else:
            for colname,col in output.iteritems():
                output[col] = SimpleImputer(missing_values=np.nan, stra

        return output

    def fit_transform(self,X,y=None):
        return self.fit(X,y).transform(X)
```

In [ ]:
```

```

In [324]:
```
#MultiColumnSimpleImputer(df.columns).fit_transform(df)

```

In [304]:
```
from sklearn.impute import SimpleImputer
imp_mean = SimpleImputer(missing_values=np.nan, strategy='mean')
df_transform = imp_mean.fit_transform(df)
#df
```

In [186]:
```
df_transformed = pd.DataFrame(df_transform, columns = df.columns)
df_transformed.head()
```

Out[186]:

	Country	Year	Status	Life expectancy	Adult Mortality	infant deaths	Alcohol	percentage expenditure	Hepatitis B	Measles
0	0.0	2015.0	1.0	65.0	263.0	62.0	0.01	71.279624	65.0	1154.0
1	0.0	2014.0	1.0	59.9	271.0	64.0	0.01	73.523582	62.0	492.0
2	0.0	2013.0	1.0	59.9	268.0	66.0	0.01	73.219243	64.0	430.0
3	0.0	2012.0	1.0	59.5	272.0	69.0	0.01	78.184215	67.0	2787.0
4	0.0	2011.0	1.0	59.2	275.0	71.0	0.01	7.097109	68.0	3013.0

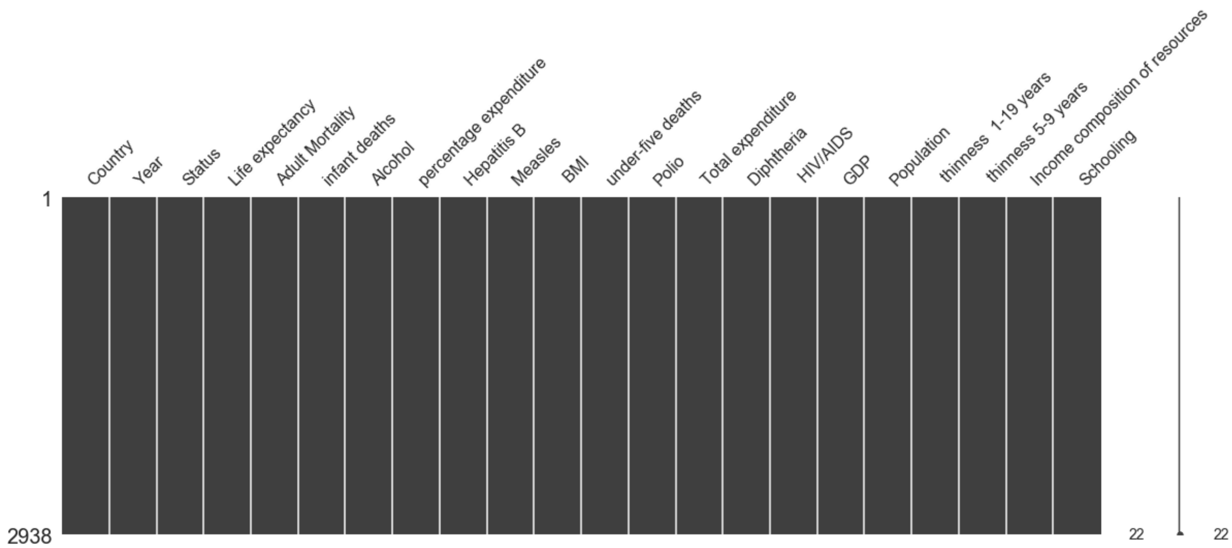
5 rows × 22 columns

In [ ]:
```

```

In [327]:

```
mno.matrix(df_transformed, figsize = (20, 6))

```

Out[327]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2afa9c50>



In [329]:

```
df_transformed.columns
```

Out[329]:

```
Index(['Country', 'Year', 'Status', 'Life expectancy ', 'Adult Mortalit
y',
       'infant deaths', 'Alcohol', 'percentage expenditure', 'Hepatitis
B',
       'Measles ', ' BMI ', 'under-five deaths ', 'Polio', 'Total expendi
ture',
       'Diphtheria ', ' HIV/AIDS', 'GDP', 'Population',
       ' thinness  1-19 years', ' thinness 5-9 years',
       'Income composition of resources', 'Schooling'],
      dtype='object')
```

In [341]:

```
from sklearn.model_selection import train_test_split
X = df_transformed.drop(['Life expectancy '], axis=1)
y = df_transformed['Life expectancy ']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
```

In [351]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, r
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

In [398]:

```
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
#rf_random = RandomizedSearchCV(estimator = rf, param_distributions = r
# Fit the random search model
rf.fit(X_train, y_train)
```

Out[398]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [404]:

```
new_ans = rf.predict(X_test)
```

In [414]:

```
import graphviz
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import pydot

from IPython.display import display
#forest_clf = RandomForestClassifier()
#forest_clf.fit(X_train, y_train)
tree.export_graphviz(rf.estimators_[0], out_file='tree_from_forest.dot'
(graph,) = pydot.graph_from_dot_file('tree_from_forest.dot')
graph.write_png('tree_from_forest.png')

```

In [431]:

```
#Let’s go back to our hypothetical medication study. Suppose the hypoth

#If the medicine has no effect in the population as a whole, 3% of stud
```

In [422]:
```python

import numpy as np
import statsmodels.api as sm
import matplotlib.pyplot as plt
from statsmodels.sandbox.regression.predstd import wls_prediction_std
```

In [430]:
```python
len(X_train['Country'])

```

Out[430]: 1968

In [423]:
```
model = sm.OLS(y_train, X_train)
results = model.fit()
print(results.summary())
```

```
                                      OLS Regression Results
==========================================================================================
==============
Dep. Variable:       Life expectancy    R-squared (uncentered):
0.997
Model:                           OLS    Adj. R-squared (uncentered):
0.997
Method:                Least Squares    F-statistic:
2.764e+04
Date:               Sun, 08 Dec 2019    Prob (F-statistic):
0.00
Time:                       23:48:18    Log-Likelihood:
-5546.9
No. Observations:               1968    AIC:
1.114e+04
Df Residuals:                   1947    BIC:
1.125e+04
Df Model:                         21
Covariance Type:           nonrobust
==========================================================================================
==========================
                                      coef    std err          t      P>|
t|      [0.025      0.975]
------------------------------------------------------------------------------------------
--------------------------
Country                             0.0032      0.002      1.955      0.0
51   -9.78e-06       0.006
Year                                0.0283      0.000     68.254      0.0
00       0.028       0.029
Status                             -1.6003      0.328     -4.880      0.0
00      -2.244      -0.957
Adult Mortality                    -0.0198      0.001    -19.913      0.0
00      -0.022      -0.018
infant deaths                       0.1009      0.010      9.663      0.0
00       0.080       0.121
Alcohol                             0.0671      0.032      2.106      0.0
35       0.005       0.130
percentage expenditure              0.0001      0.000      1.280      0.2
01   -6.96e-05       0.000
Hepatitis B                        -0.0146      0.005     -3.044      0.0
02      -0.024      -0.005
Measles                         -2.515e-05   9.12e-06     -2.758      0.0
06    -4.3e-05   -7.26e-06
 BMI                                0.0407      0.006      6.690      0.0
00       0.029       0.053
under-five deaths                  -0.0746      0.008     -9.780      0.0
00      -0.090      -0.060
Polio                               0.0249      0.006      4.390      0.0
00       0.014       0.036
Total expenditure                   0.0218      0.041      0.525      0.5
99      -0.060       0.103
Diphtheria                          0.0374      0.006      6.376      0.0
00       0.026       0.049
```

```
 HIV/AIDS                             -0.4772      0.021    -22.266      0.0
00      -0.519      -0.435
GDP                                  2.69e-05   1.54e-05      1.751      0.0
80   -3.23e-06     5.7e-05
Population                         -6.654e-10   2.44e-09     -0.273      0.7
85   -5.45e-09    4.12e-09
 thinness  1-19 years                 -0.0796      0.060     -1.335      0.1
82      -0.197       0.037
 thinness 5-9 years                    0.0037      0.059      0.063      0.9
50      -0.112       0.120
Income composition of resources        6.1349      0.802      7.645      0.0
00       4.561       7.709
Schooling                              0.6616      0.052     12.604      0.0
00       0.559       0.765
==========================================================================
=====
Omnibus:                       87.099   Durbin-Watson:
1.999
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                 23
6.355
Skew:                          -0.184   Prob(JB):                        4.7
5e-52
Kurtosis:                       4.657   Cond. No.                        3.9
2e+08
==========================================================================
=====

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 3.92e+08. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [419]:
```
print('Parameters: ', results.params)
print('R2: ', results.rsquared)

```

```
Parameters:  Country                                           3.228836e-03
Year                                 2.834522e-02
Status                              -1.600320e+00
Adult Mortality                     -1.982795e-02
infant deaths                        1.009259e-01
Alcohol                              6.707656e-02
percentage expenditure               1.308046e-04
Hepatitis B                         -1.458378e-02
Measles                             -2.514781e-05
 BMI                                 4.070490e-02
under-five deaths                   -7.461739e-02
Polio                                2.485544e-02
Total expenditure                    2.179709e-02
Diphtheria                           3.737595e-02
 HIV/AIDS                           -4.772125e-01
GDP                                  2.690468e-05
Population                          -6.654124e-10
 thinness  1-19 years               -7.962809e-02
 thinness 5-9 years                  3.741886e-03
Income composition of resources      6.134878e+00
Schooling                            6.615710e-01
dtype: float64
R2:  0.9966563487669086
```

In [433]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
#>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
#>>> # y = 1 * x_0 + 2 * x_1 + 3
#>>> y = np.dot(X, np.array([1, 2])) + 3
reg = LinearRegression().fit(X_train, y_train)
#reg.score(X, y)
```

In [434]:
```
reg.score(X, y)
```

Out[434]:
```
0.8197513550724889
```

In [442]:
```
reg.coef_
```

Out[442]:
```
array([ 3.21119912e-03,  2.00132715e-03, -1.57661894e+00, -1.97239820e-0
2,
        1.00552922e-01,  6.16509079e-02,  1.18951459e-04, -1.44508920e-0
2,
       -2.56680107e-05,  4.06253988e-02, -7.43388467e-02,  2.46560630e-0
2,
        2.73154781e-02,  3.75368626e-02, -4.80112235e-01,  2.88480648e-0
5,
       -6.30147861e-10, -7.79598914e-02,  2.81391552e-03,  6.26047718e+0
0,
        6.67670054e-01])
```

In [441]:
```
reg.intercept_
```
Out[441]: 52.68181514202163

In [450]:
```
y_pred = reg.predict(X_test)

```

In [458]:
```
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute
from math import sqrt
```

In [482]:
```
def print_score(y_test, y_pred):
    print(" mean_absolute_error   =    {} ".format(round(mean_absolute_e
    print(" mean_squared_error    =    {} ".format(round(mean_squared_er
    print(" root_mean_squared_error = {} ".format(round(sqrt(mean_squar
    print(" r2_score =                {} ".format(round(r2_score(y_test

```

In [459]:
```
mean_squared_error?
```

In [510]:
```
print_score(y_test, y_pred)
```
```
mean_absolute_error   =    2.937
mean_squared_error    =    16.022
root_mean_squared_error = 4.003
r2_score =                 0.826
```

In [466]:
```
round(2.7817191, ndigits= 3)
```
Out[466]: 2.782

In [583]:
```
from sklearn import linear_model
clf = linear_model.Lasso(alpha=1)
clf.fit(X_train, y_train)
```
Out[583]:
```
Lasso(alpha=1, copy_X=True, fit_intercept=True, max_iter=1000, normalize=
False,
      positive=False, precompute=False, random_state=None, selection='cyc
lic',
      tol=0.0001, warm_start=False)
```

In [ ]:
```

```

In [579]:
```
print(clf.coef_)
```
```
[ 2.60927399e-03  0.00000000e+00 -0.00000000e+00 -2.32146112e-02
  8.85865351e-02  9.56471060e-02  1.66652748e-04 -1.25303024e-02
 -3.06587413e-05  5.35835119e-02 -6.58977372e-02  2.52542915e-02
  0.00000000e+00  4.32485363e-02 -4.34057521e-01  5.32035204e-05
  1.19951964e-10 -3.94846945e-02 -0.00000000e+00  0.00000000e+00
  8.27412021e-01]
```

In [580]:
```
print(clf.intercept_)
```

```
56.434209676507805
```

In [581]:
```
y_pred_l = clf.predict(X_test)
```

In [582]:
```
print_score(y_test, y_pred_l)
```

```
mean_absolute_error  =    2.99
mean_squared_error   =    16.751
root_mean_squared_error = 4.093
r2_score =                0.818
```

In [600]:
```
from sklearn import linear_model
#sklearn.linear_model.Ridge
clf = linear_model.Ridge(alpha=0.01)
clf.fit(X_train, y_train)
```

Out[600]:
```
Ridge(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [601]:
```
print(clf.coef_)
```

```
[ 3.21118420e-03  2.00926971e-03 -1.57658134e+00 -1.97243352e-02
  1.00556085e-01  6.16596866e-02  1.18940852e-04 -1.44512583e-02
 -2.56688588e-05  4.06264691e-02 -7.43410472e-02  2.46558922e-02
  2.73047814e-02  3.75380577e-02 -4.80113387e-01  2.88533299e-05
 -6.30161628e-10 -7.79654641e-02  2.81619556e-03  6.25803872e+00
  6.67765661e-01]
```

In [602]:
```
y_pred_l = clf.predict(X_test)
```

In [603]:
```
print_score(y_test, y_pred_l)
```

```
mean_absolute_error  =    2.937
mean_squared_error   =    16.022
root_mean_squared_error = 4.003
r2_score =                0.826
```

In [604]:
```
from sklearn.linear_model import ElasticNet

```

In [605]:
```
regr = ElasticNet(random_state=0)
regr.fit(X_train, y_train)

```

Out[605]:
```
ElasticNet(alpha=1.0, copy_X=True, fit_intercept=True, l1_ratio=0.5,
           max_iter=1000, normalize=False, positive=False, precompute=Fal
se,
           random_state=0, selection='cyclic', tol=0.0001, warm_start=Fal
se)
```

In [610]:
```
y_pred_r = regr.predict(X_test)
```

In [607]:
```
print(regr.coef_)

```

```
[ 3.07948392e-03  8.03124380e-03 -0.00000000e+00 -2.27788706e-02
  9.99014440e-02  1.28906758e-01  1.51438845e-04 -1.40664886e-02
 -2.97554470e-05  5.10817223e-02 -7.38323401e-02  2.57368590e-02
  0.00000000e+00  4.31514582e-02 -4.46882496e-01  5.28701233e-05
 -3.18901057e-10 -7.04940446e-02 -0.00000000e+00  0.00000000e+00
  8.08000374e-01]
```

In [608]:
```

print(regr.intercept_)

```

```
40.646272811209904
```

In [612]:
```
print_score(y_test, y_pred_r)
```

```
 mean_absolute_error  =    2.987
 mean_squared_error   =    16.614
 root_mean_squared_error = 4.076
 r2_score =                0.819
```

In [637]:
```
from sklearn.tree import DecisionTreeRegressor

# create a regressor object
regressor = DecisionTreeRegressor(random_state = 0, max_depth = 6)

# fit the regressor with X and Y data
regressor.fit(X_train, y_train)
```

Out[637]:
```
DecisionTreeRegressor(criterion='mse', max_depth=6, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=0, splitter='best')
```

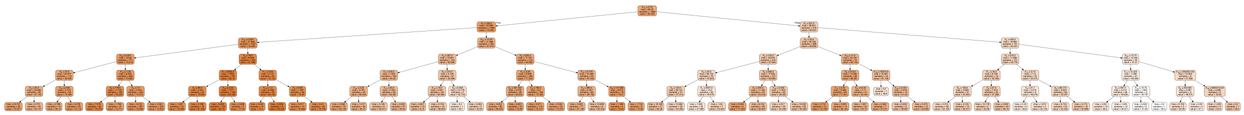
In [638]:
```
y_pred_tree = regressor.predict(X_test)
```

In [639]:
```
print_score(y_test, y_pred_tree)
```

```
 mean_absolute_error  =    1.92
 mean_squared_error   =    7.032
 root_mean_squared_error = 2.652
 r2_score =                0.923
```

In [640]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(regressor, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

Out[640]:



In [642]:
```
graph.write_pdf("iris.pdf")

```

Out[642]:
```
True
```

In [644]:
```
#Image(graph.create_png())
```

In [645]:
```
from sklearn.ensemble import ExtraTreesRegressor
```

In [682]:
```
regressor = ExtraTreesRegressor(random_state = 0, max_depth = 10)

# fit the regressor with X and Y data
regressor.fit(X_train, y_train)
```

Out[682]:
```
ExtraTreesRegressor(bootstrap=False, criterion='mse', max_depth=10,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs
=None,
                    oob_score=False, random_state=0, verbose=0,
                    warm_start=False)
```

In [683]:
```
y_pred_extra = regressor.predict(X_test)
```

In [684]:
```
print_score(y_test, y_pred_extra)
```

```
mean_absolute_error  =    1.311
mean_squared_error   =    3.751
root_mean_squared_error = 1.937
r2_score =                0.959
```

In [685]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [690]:
```
regressor = GradientBoostingRegressor(random_state = 0, max_depth = 11)

# fit the regressor with X and Y data
regressor.fit(X_train, y_train)
```

Out[690]:
```
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
                          learning_rate=0.1, loss='ls', max_depth=11,
                          max_features=None, max_leaf_nodes=None,
                          min_impurity_decrease=0.0, min_impurity_split=N
one,
                          min_samples_leaf=1, min_samples_split=2,
                          min_weight_fraction_leaf=0.0, n_estimators=100,
                          n_iter_no_change=None, presort='auto', random_s
tate=0,
                          subsample=1.0, tol=0.0001, validation_fraction=
0.1,
                          verbose=0, warm_start=False)
```

In [691]:
```
y_pred_xg = regressor.predict(X_test)
```

In [692]:
```
print_score(y_test, y_pred_extra)
```

```
mean_absolute_error  =    1.058
mean_squared_error   =    2.939
root_mean_squared_error = 1.714
r2_score =                0.968
```

In [ ]:
```

```