Data Preprocessing

In [384]:

```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [385]:

```
# Importing the dataset
df = pd.read_csv('datasets_happiness_2019_.csv', sep=',')
```

In [386]:

```
df
```

Out[386]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
0	0.986	1.340	7.769	1.587	0.596	0.393	0.153
1	0.996	1.383	7.600	1.573	0.592	0.410	0.252
2	1.028	1.488	7.554	1.582	0.603	0.341	0.271
3	1.026	1.380	7.494	1.624	0.591	0.118	0.354
4	0.999	1.396	7.488	1.522	0.557	0.298	0.322
...	...	...	...	...	...	...	...
151	0.614	0.359	3.334	0.711	0.555	0.411	0.217
152	0.499	0.476	3.231	0.885	0.417	0.147	0.276
153	0.361	0.350	3.203	0.517	0.000	0.025	0.158
154	0.105	0.026	3.083	0.000	0.225	0.035	0.235
155	0.295	0.306	2.853	0.575	0.010	0.091	0.202

156 rows × 7 columns

In [387]:

```
df.describe()
# Критических выбросов не наблюдается
```

Out[387]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
count	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000
mean	0.725244	0.905147	5.407096	1.208814	0.392571	0.110603	0.184846
std	0.242124	0.398389	1.113120	0.299191	0.143289	0.094538	0.095254
min	0.000000	0.000000	2.853000	0.000000	0.000000	0.000000	0.000000
25%	0.547750	0.602750	4.544500	1.055750	0.308000	0.047000	0.108750
50%	0.789000	0.960000	5.379500	1.271500	0.417000	0.085500	0.177500
75%	0.881750	1.232500	6.184500	1.452500	0.507250	0.141250	0.248250
max	1.141000	1.684000	7.769000	1.624000	0.631000	0.453000	0.566000

In [388]:

```
# mean()-3*std
# Let's check how much the data are spread out from the mean.

mean_Score = np.mean(df['Score'], axis=0)
sd_Score = np.std(df['Score'], axis=0)

mean_GDP_per_capita = np.mean(df['GDP per capita'], axis=0)
sd_GDP_per_capita = np.std(df['GDP per capita'], axis=0)

mean_Social_support = np.mean(df['Social support'], axis=0)
sd_Social_support = np.std(df['Social support'], axis=0)

mean_Healthy_life_expectancy = np.mean(df['Healthy life expectancy'], axis=0)
sd_Healthy_life_expectancy = np.std(df['Healthy life expectancy'], axis=0)

mean_Freedom_to_make_life_choices = np.mean(df['Freedom to make life choices'], axis=0)
sd_Freedom_to_make_life_choices = np.std(df['Freedom to make life choices'], axis=0)

mean_Generosity = np.mean(df['Generosity'], axis=0)
sd_Generosity = np.std(df['Generosity'], axis=0)

mean_Perceptions_of_corruption = np.mean(df['Perceptions of corruption'], axis=0)
sd_Perceptions_of_corruption = np.std(df['Perceptions of corruption'], axis=0)

counter_Score = 0
counter_GDP_per_capita = 0
counter_Social_support = 0
counter_Healthy_life_expectancy = 0
counter_Freedom_to_make_life_choices = 0
counter_Generosity = 0
counter_Perceptions_of_corruption = 0

for Score, GDP_per_capita, Social_support, Healthy_life_expectancy, Freedom_to_make_life_c
hoices, Generosity, Perceptions_of_corruption in zip( df['Score'], df['GDP per capita'], d
f['Social support'], df['Healthy life expectancy'], df['Freedom to make life choices'], df
['Generosity'], df['Perceptions of corruption']):
    if not mean_Score - 3*sd_Score <= Score <= mean_Score + 3*sd_Score:
        counter_Score += 1
    if not mean_GDP_per_capita - 3*sd_GDP_per_capita <= counter_GDP_per_capita <= mean_GDP
_per_capita + 3*sd_GDP_per_capita:
        counter_GDP_per_capita += 1
    if not mean_Social_support - 3*sd_Social_support <= counter_Social_support <= mean_Soc
ial_support + 3*sd_Social_support:
        counter_Social_support += 1
    if not mean_Healthy_life_expectancy - 3*sd_Healthy_life_expectancy <= counter_Healthy_
life_expectancy <= mean_Healthy_life_expectancy + 3*sd_Healthy_life_expectancy:
        counter_Healthy_life_expectancy += 1
    if not mean_Freedom_to_make_life_choices - 3*sd_Freedom_to_make_life_choices <= counte
r_Freedom_to_make_life_choices <= mean_Freedom_to_make_life_choices + 3*sd_Freedom_to_make
_life_choices:
        counter_Freedom_to_make_life_choices += 1
    if not mean_Generosity - 3*sd_Generosity <= counter_Generosity <= mean_Generosity + 3*
sd_Generosity:
        counter_Generosity += 1
    if not mean_Perceptions_of_corruption - 3*sd_Perceptions_of_corruption <= counter_Perc
```

```
eptions_of_corruption <= mean_Perceptions_of_corruption + 3*sd_Perceptions_of_corruption:
        counter_Perceptions_of_corruption += 1

counter_dicts = {
                'counter_Score': counter_Score,
                'counter_GDP_per_capita': counter_GDP_per_capita,
                'counter_Social_support': counter_Social_support,
                'counter_Healthy_life_expectancy': counter_Healthy_life_expectancy,
                'counter_Freedom_to_make_life_choices': counter_Freedom_to_make_life_choic
es,
                'counter_Generosity': counter_Generosity,
                'counter_Perceptions_of_corruption': counter_Perceptions_of_corruption}
print(counter_dicts)
```

```
{'counter_Score': 0, 'counter_GDP_per_capita': 0, 'counter_Social_support':
1, 'counter_Healthy_life_expectancy': 1, 'counter_Freedom_to_make_life_choice
s': 0, 'counter_Generosity': 0, 'counter_Perceptions_of_corruption': 0}
```

Как видим, за рамки 3 сигм выходит лишь по одному элементу из столбцов Social_support и Healthy_life_expectancy. Избавимся от них.

In [389]:

```
# Function Outliers
def outliers(df):
    num_var = list(df._get_numeric_data().columns)
    for col_names in num_var:
        df[col_names] = df[col_names].apply(lambda y: df[col_names].mean()-3*df[col_names]
.std()
                            if y < df[col_names].mean()-3*df[col_names].std() else y)
        df[col_names] = df[col_names].apply(lambda y: df[col_names].mean()+3*df[col_names]
.std()
                            if y > df[col_names].mean()+3*df[col_names].std() else y)
    return(df)


# Outliers
df = outliers(df)
df.describe()
```

Out[389]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
count	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000	156.000000
mean	0.725244	0.905147	5.407096	1.210809	0.392571	0.110017	0.184059
std	0.242124	0.398389	1.113120	0.292031	0.143289	0.092612	0.092501
min	0.000000	0.000000	2.853000	0.311240	0.000000	0.000000	0.000000
25%	0.547750	0.602750	4.544500	1.055750	0.308000	0.047000	0.108750
50%	0.789000	0.960000	5.379500	1.271500	0.417000	0.085500	0.177500
75%	0.881750	1.232500	6.184500	1.452500	0.507250	0.141250	0.248250
max	1.141000	1.684000	7.769000	1.624000	0.631000	0.394216	0.470609

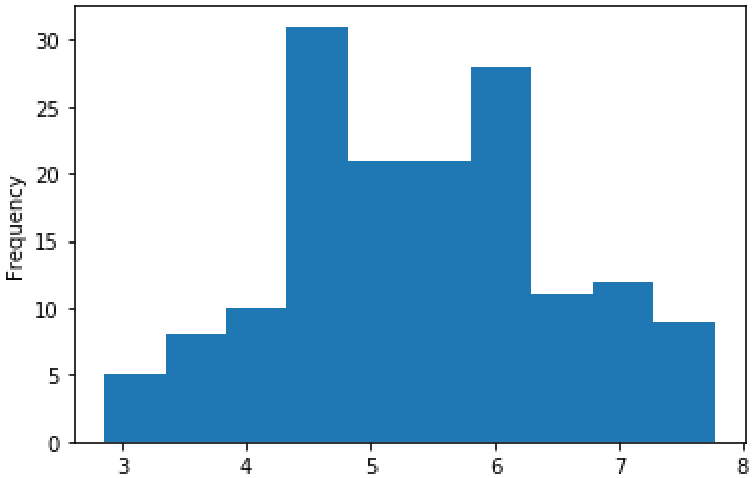
Построим гистограммы для того, чтобы посмотреть, как распределяется каждая переменная.

In [390]:

```
# Overall_rank distribution
df['Score'].plot(kind = 'hist')
```

Out[390]:

<matplotlib.axes._subplots.AxesSubplot at 0x155349d0ac0>

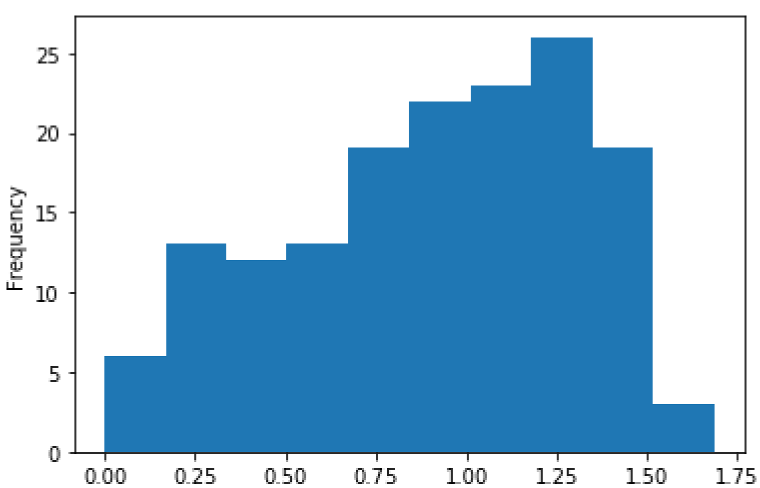


In [391]:

```
# Score distribution
# distribution = df['Score'].value_counts()
# distribution.plot(kind='bar')
df['GDP per capita'].plot(kind = 'hist')
```

Out[391]:

<matplotlib.axes._subplots.AxesSubplot at 0x15534a4bdf0>

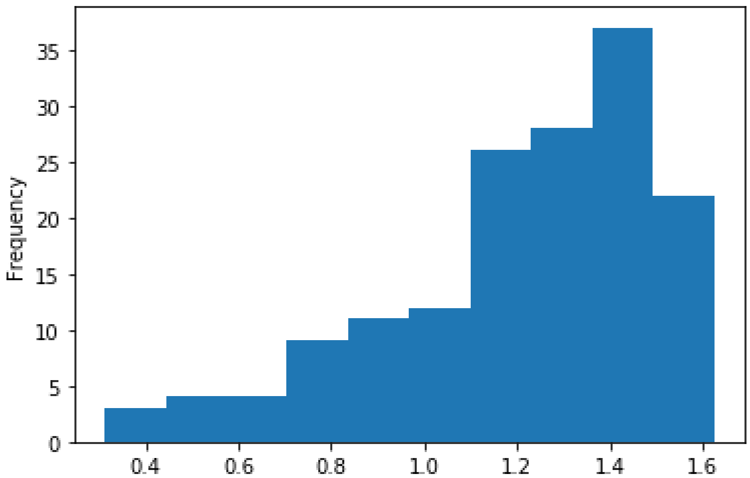


In [392]:

```
# Social support distribution
df['Social support'].plot(kind = 'hist')
```

Out[392]:

<matplotlib.axes._subplots.AxesSubplot at 0x15534ab19d0>

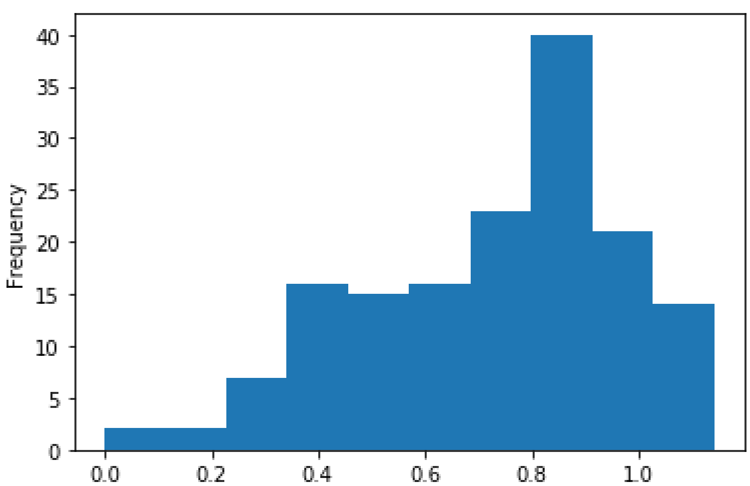


In [393]:

```
# Healthy_life_expectancy distribution
df['Healthy life expectancy'].plot(kind = 'hist')
```

Out[393]:

<matplotlib.axes._subplots.AxesSubplot at 0x15534b1a0d0>

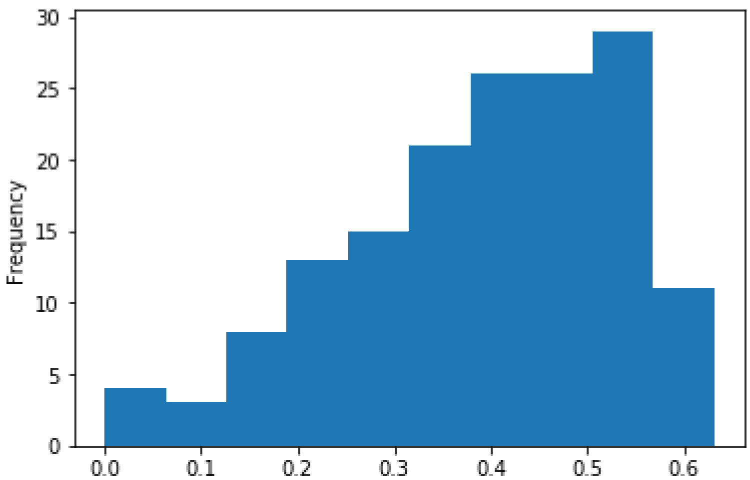


In [394]:

```
# Freedom to make life choices distribution
df['Freedom to make life choices'].plot(kind = 'hist')
```

Out[394]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15534b84f70>
```

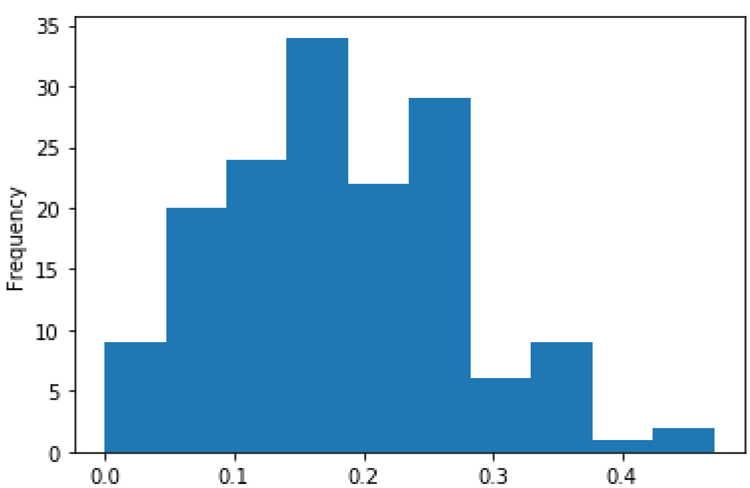


In [395]:

```
# Freedom to make life choices distribution
df['Generosity'].plot(kind = 'hist')
```

Out[395]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15534a9ecd0>
```

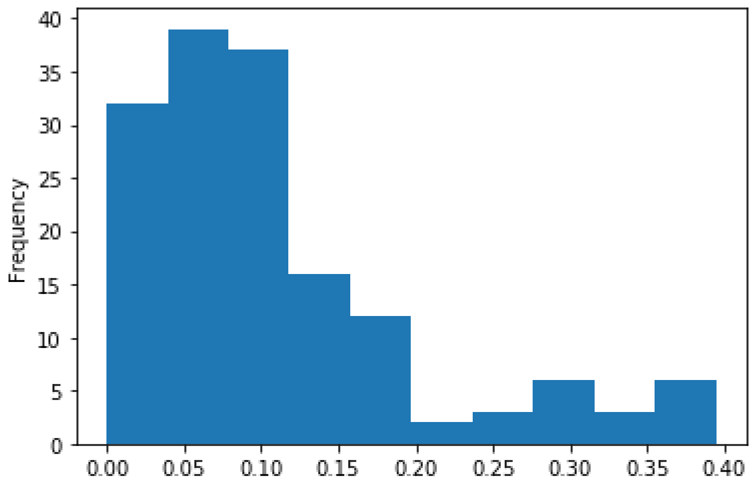


In [396]:

```
# Freedom to make life choices distribution
df['Perceptions of corruption'].plot(kind = 'hist')
```

Out[396]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x15534c51730>
```



In [397]:

```
# Проверим пропущенные данные в колонках.
df.isnull().sum()
# Таким образом у нас нет пропущенных данных.
```

Out[397]:

```
Healthy life expectancy         0
GDP per capita                  0
Score                           0
Social support                  0
Freedom to make life choices    0
Perceptions of corruption       0
Generosity                      0
dtype: int64
```

Linear Regression

In [398]:

```
df
```

Out[398]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
0	0.986	1.340	7.769	1.58700	0.596	0.393000	0.153
1	0.996	1.383	7.600	1.57300	0.592	0.394216	0.252
2	1.028	1.488	7.554	1.58200	0.603	0.341000	0.271
3	1.026	1.380	7.494	1.62400	0.591	0.118000	0.354
4	0.999	1.396	7.488	1.52200	0.557	0.298000	0.322
...	...	...	...	...	...	...	...
151	0.614	0.359	3.334	0.71100	0.555	0.394216	0.217
152	0.499	0.476	3.231	0.88500	0.417	0.147000	0.276
153	0.361	0.350	3.203	0.51700	0.000	0.025000	0.158
154	0.105	0.026	3.083	0.31124	0.225	0.035000	0.235
155	0.295	0.306	2.853	0.57500	0.010	0.091000	0.202

156 rows × 7 columns

In [399]:

```
# Cheking correlations
correlation = df.corr()
correlation.style.background_gradient(cmap='coolwarm')
```

Out[399]:

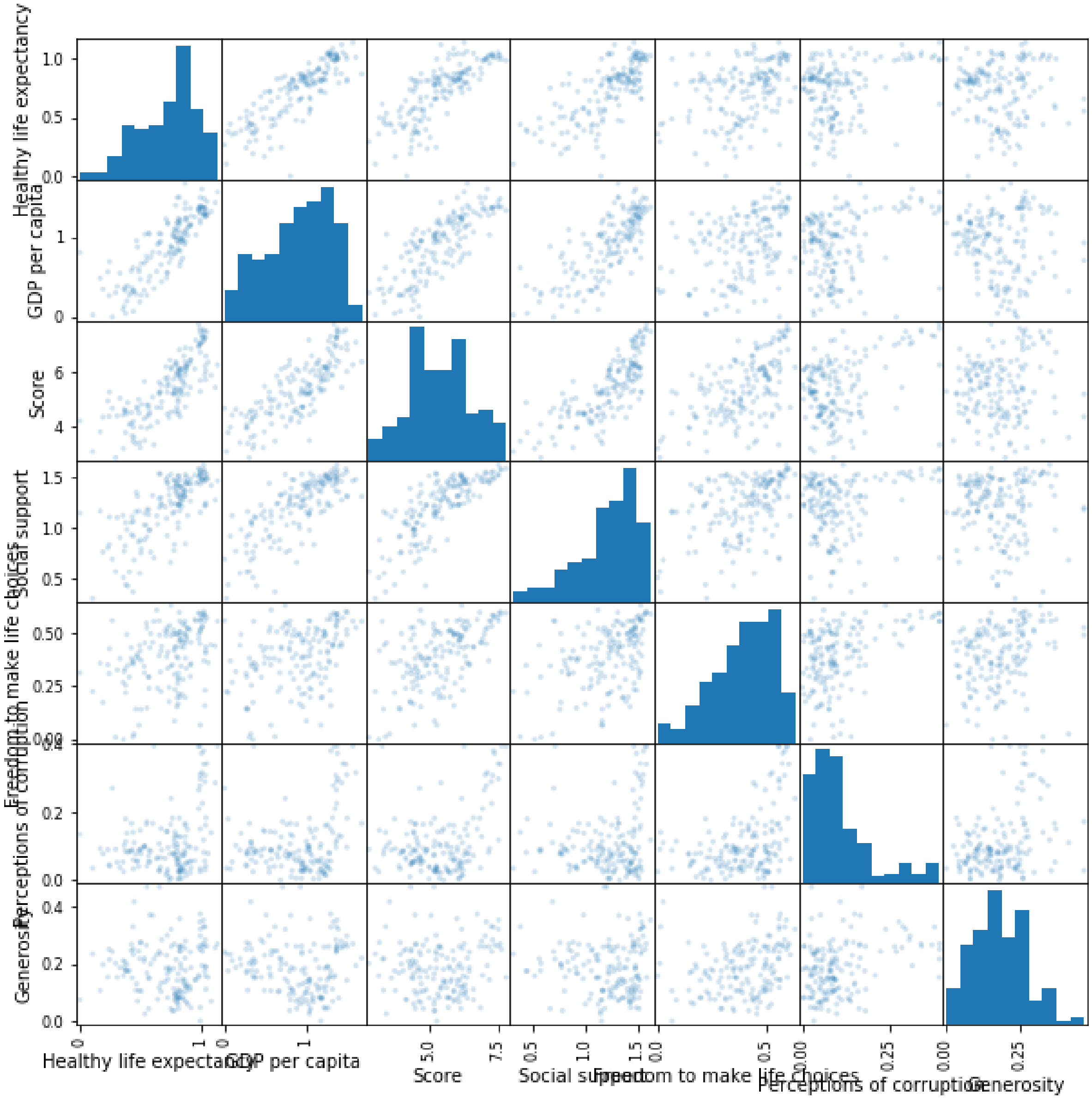
	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
Healthy life expectancy	1.000000	0.835462	0.779883	0.719026	0.390395	0.293698	-0.025196
GDP per capita	0.835462	1.000000	0.793883	0.758243	0.379079	0.298564	-0.078898
Score	0.779883	0.793883	1.000000	0.781755	0.566742	0.390497	0.084708
Social support	0.719026	0.758243	0.781755	1.000000	0.450261	0.181722	-0.046316
Freedom to make life choices	0.390395	0.379079	0.566742	0.450261	1.000000	0.440441	0.270309
Perceptions of corruption	0.293698	0.298564	0.390497	0.181722	0.440441	1.000000	0.335468
Generosity	-0.025196	-0.078898	0.084708	-0.046316	0.270309	0.335468	1.000000

In [400]:

```
# Как мы видим, корреляция с y (Healthy life expectancy) самая значительная с GDP per capi
ta (0.835462), Score (0.779883) и с Social suppor (0.718832).
```

In [401]:

```
from pandas.plotting import scatter_matrix
scatter_matrix(df, alpha=0.2, figsize=(10, 10))
plt.show()
```


Healthy life expectancy
GDP per capita
Score
Social support
Freedom to make life choices
Perceptions of corruption
Generosity

In [402]:

```
# Splitting the dataset into the Training set and Test set
X = df.iloc[:, 1:6].values
y = df.iloc[:, 0].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [403]:

```
# Построим линейную регрессию с GDP per capita (0.835462) в качестве X.
```

In [405]:

```
# Fitting Simple Linear Regression to the Training set (GDP per capita)
from sklearn.linear_model import LinearRegression
sr = LinearRegression().fit(X_train[:, 1:2], y_train)
```

In [406]:

```
# Getting parameters
sr.coef_, sr.intercept_
```

Out[406]:

```
(array([0.17188042]), -0.20325738118992775)
```

In [407]:

```
# Predicting the Test set results
y_pred = sr.predict(X_test[:, 1:2])
```

In [408]:

```
# Coefficient of determination R^2
sr.score(X_train[:, 1:2], y_train), sr.score(X_test[:, 1:2], y_test)
```

Out[408]:

```
(0.6007918214228924, 0.6447212155182563)
```

In [409]:

```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, sr.predict(X_train[:, 1:2])), mean_squared_error(y_test, y_pre
d)
```

Out[409]:

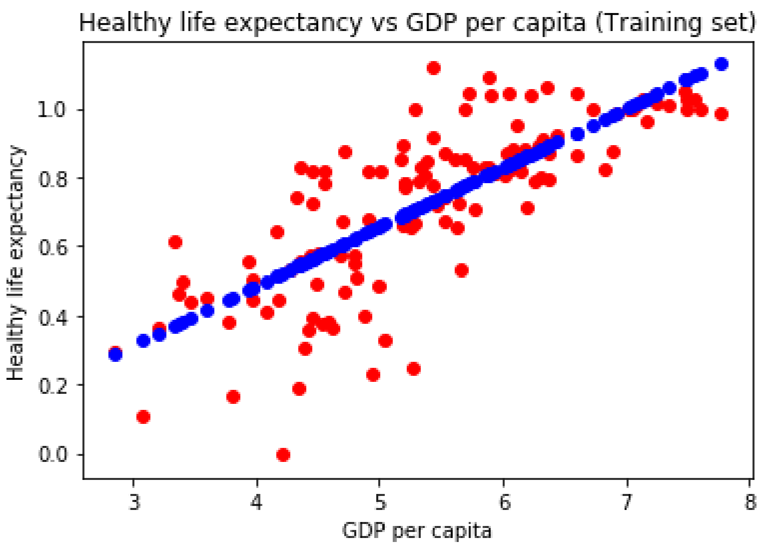
```
(0.02440447080805655, 0.01671709779534196)
```

In [410]:

```
# Исходя из R^2 и MSE делаем вывод, что модель адеватна и ее нельзя использовать для прогн
озирования.
```

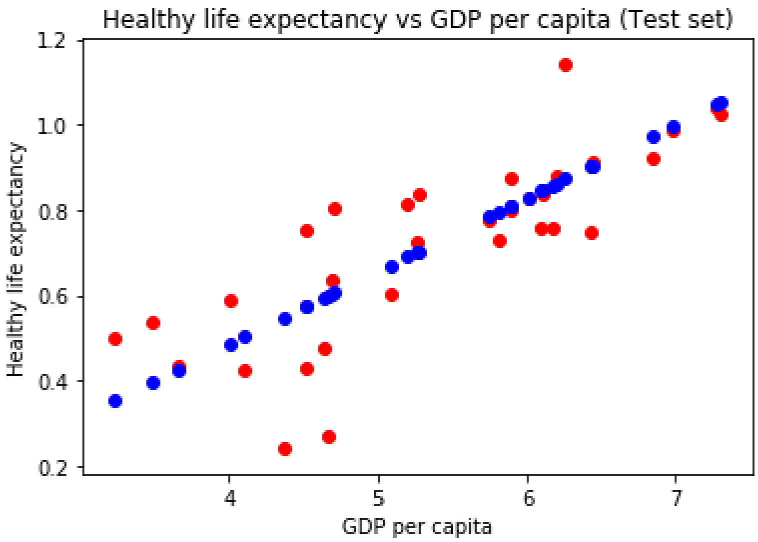
In [411]:

```
# Visualising the Training set results
plt.scatter(X_train[:,1], y_train, color = 'red')
plt.plot(X_train[:,1], sr.predict(X_train[:, 1:2]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [412]:

```
# Visualising the Test set results
plt.scatter(X_test[:,1], y_test, color = 'red')
plt.plot(X_test[:,1], sr.predict(X_test[:, 1:2]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [413]:

```
# Fitting Multiple Linear Regression to the Training set
# Построим линейную регрессию используя три самые коррелируемые переменные (GDP per capita
(0.835462), Score (0.779883) и Social suppor (0.718832)).
from sklearn.linear_model import LinearRegression
mr = LinearRegression().fit(X_train[:, 0:3], y_train)
```

In [414]:

```
# Getting parameters
mr.coef_, mr.intercept_
```

Out[414]:

```
(array([0.33342874, 0.05459521, 0.08466052]), 0.026469686939751381)
```

In [415]:

```
# Predicting the Test set results
y_pred = mr.predict(X_test[:, 0:3])
```

In [416]:

```
# Coefficient of determination R^2
mr.score(X_train[:, 0:3], y_train), mr.score(X_test[:, 0:3], y_test)
```

Out[416]:

```
(0.7151031441860636, 0.8468652866187845)
```

In [417]:

```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, mr.predict(X_train[:,0:3])), mean_squared_error(y_test, y_pred
)
```

Out[417]:

```
(0.017416369138027004, 0.007205518852440761)
```

In [418]:

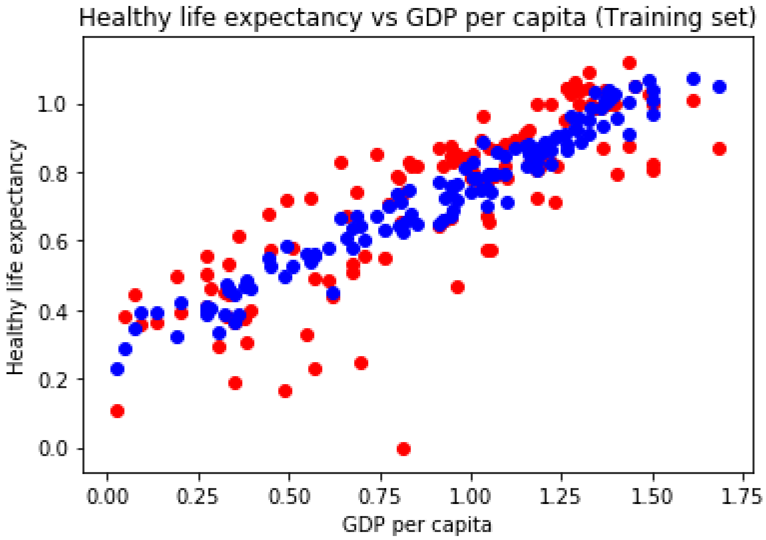
```
# !pip install statsmodels
# p-values
import statsmodels.api as sm
X = sm.add_constant(X_train)
mr1 = sm.OLS(y_train, X).fit()
mr1.pvalues
#mr1.summary()
```

Out[418]:

```
array([7.51636986e-01, 5.12616046e-08, 2.06634137e-02, 2.49878317e-01,
       8.01081761e-01, 9.72690588e-01])
```

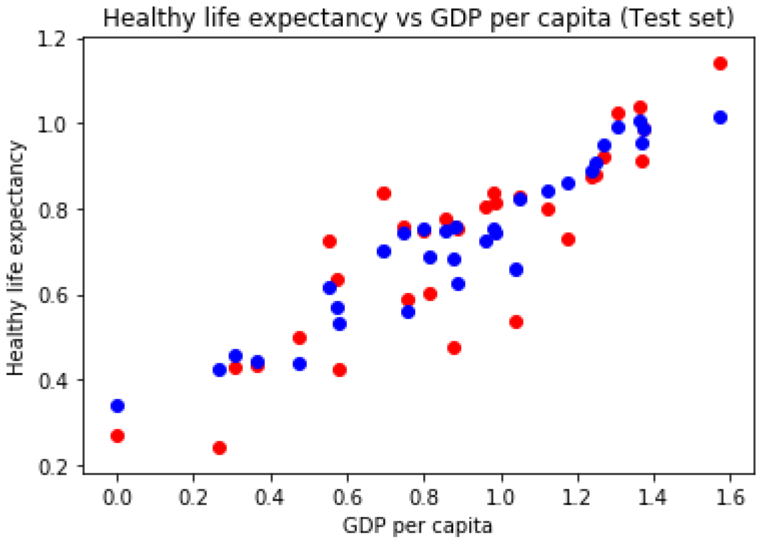
In [419]:

```
# Visualising the Training set results
plt.scatter(X_train[:,0], y_train, color = 'red')
plt.plot(X_train[:,0], mr.predict(X_train[:, 0:3]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [420]:

```
# Visualising the Test set results
plt.scatter(X_test[:,0], y_test, color = 'red')
plt.plot(X_test[:,0], mr.predict(X_test[:,0:3]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [421]:

```
# Fitting Polynomial Regression to the dataset (GDP per capita)
from sklearn.preprocessing import PolynomialFeatures
X_train_p = PolynomialFeatures().fit_transform(X_train[:, 0:1])
X_test_p = PolynomialFeatures().fit_transform(X_test[:, 0:1])
pr = LinearRegression().fit(X_train_p[:,0:], y_train)
```

In [422]:

```
# Getting parameters
pr.coef_, pr.intercept_
```

Out[422]:

```
(array([ 0.        ,  0.5386057 , -0.02023232]), 0.2566555083985745)
```

In [423]:

```
# Predicting the Test set results
y_pred = pr.predict(X_test_p[:,0:])
```

In [424]:

```
# Coefficient of determination R^2
pr.score(X_train_p[:,0:], y_train), pr.score(X_test_p[:,0:], y_test)
```

Out[424]:

```
(0.6832933964115014, 0.7725186782820886)
```

In [425]:

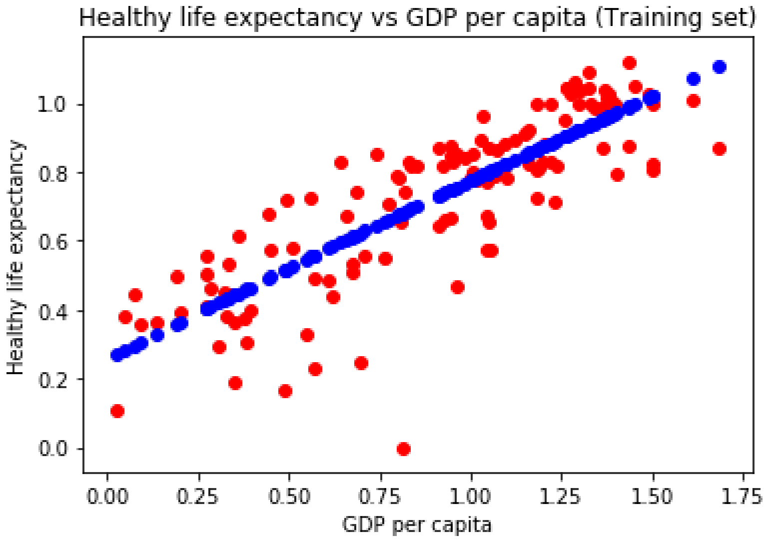
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, pr.predict(X_train_p[:,0:])), mean_squared_error(y_test, y_pre
d)
```

Out[425]:

```
(0.019360968729505563, 0.010703784374063528)
```

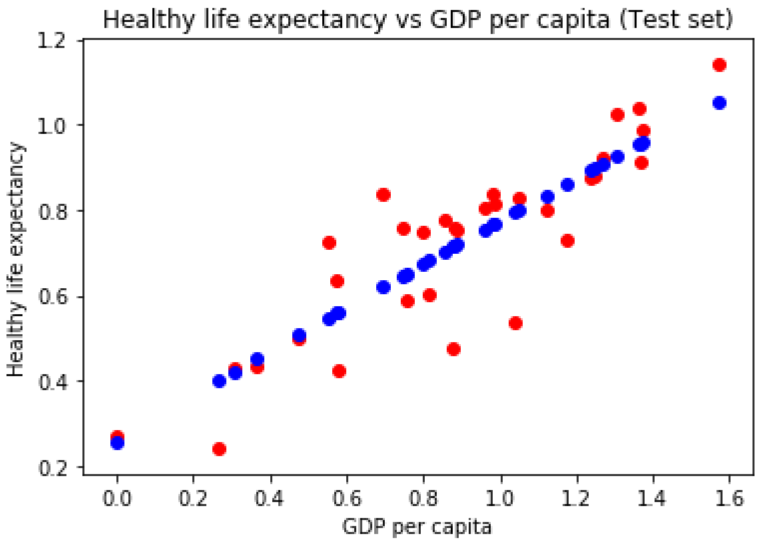
In [426]:

```
# Visualising the Training set results
plt.scatter(X_train[:,0], y_train, color = 'red')
plt.plot(X_train[:,0], pr.predict(X_train_p[:,0:]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [427]:

```
# Visualising the Test set results
plt.scatter(X_test[:,0], y_test, color = 'red')
plt.plot(X_test[:,0], pr.predict(X_test_p[:,0:]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [428]:

```
# Backward Elimination with p-values
import statsmodels.api as sm
def backwardElimination(x, sl):
    numVars = len(x[0])
    for i in range(0, numVars):
        regressor_OLS = sm.OLS(y, x).fit()
        maxVar = max(regressor_OLS.pvalues).astype(float)
        if maxVar > sl:
            for j in range(0, numVars - i):
                if (regressor_OLS.pvalues[j].astype(float) == maxVar):
                    x = np.delete(x, j, 1)
    regressor_OLS.summary()
    return x

SL = 0.05
X_opt = X_train[:, [0, 1, 2, 3, 4]]
y = y_train
X_Modeled = backwardElimination(X_opt, SL)
```

In [429]:

```
X_Modeled
```

Out[429]:

```
array([[0.82 , 4.332],
       [0.611, 4.996],
       [1.07 , 6.595],
       [1.067, 5.323],
       [0.393, 4.883],
       [0.673, 4.812],
       [1.029, 5.191],
       [1.403, 6.375],
       [0.657, 4.696],
       [0.275, 4.085],
       [1.092, 5.011],
       [1.433, 6.892],
       [1.201, 5.758],
       [1.149, 6.321],
       [1.044, 5.525],
       [0.776, 5.779],
       [1.488, 7.554],
       [0.274, 3.933],
       [1.258, 6.118],
       [0.677, 5.653],
       [1.372, 7.228],
       [0.35 , 3.203],
       [0.949, 4.366],
       [0.287, 3.38 ],
       [0.489, 3.802],
       [1.263, 5.718],
       [1.043, 4.437],
       [1.043, 5.208],
       [0.274, 3.973],
       [0.549, 5.044],
       [0.35 , 4.35 ],
       [0.493, 5.467],
       [0.026, 3.083],
       [0.512, 4.509],
       [1.5  , 6.021],
       [0.741, 5.175],
       [0.96 , 5.697],
       [0.569, 4.944],
       [0.837, 4.906],
       [1.004, 5.603],
       [0.912, 6.028],
       [1.263, 6.046],
       [1.609, 7.09 ],
       [0.921, 4.461],
       [0.45 , 4.681],
       [1.503, 6.825],
       [1.499, 7.021],
       [1.183, 5.373],
       [1.187, 5.94 ],
       [1.015, 5.425],
       [1.38 , 7.494],
       [0.306, 2.853],
       [1.052, 5.247],
       [1.387, 7.343],
       [0.807, 5.631],
```

```
       [0.57 , 4.49 ],
       [0.696, 5.265],
       [0.947, 4.719],
       [1.276, 7.139],
       [0.138, 4.628],
       [1.333, 7.054],
       [0.332, 4.189],
       [0.931, 5.192],
       [0.336, 4.286],
       [1.092, 6.086],
       [1.034, 7.167],
       [0.204, 4.466],
       [1.327, 5.886],
       [1.238, 6.149],
       [1.221, 5.693],
       [1.383, 7.6  ],
       [0.094, 4.418],
       [0.046, 3.775],
       [0.985, 6.125],
       [0.96 , 4.722],
       [1.396, 7.488],
       [0.811, 4.212],
       [1.356, 6.923],
       [0.764, 4.796],
       [1.231, 6.192],
       [1.452, 7.48 ],
       [1.301, 5.895],
       [0.191, 3.41 ],
       [0.562, 4.456],
       [1.34 , 7.769],
       [0.794, 6.253],
       [1.684, 6.374],
       [0.831, 5.89 ],
       [0.331, 4.587],
       [1.438, 5.43 ],
       [1.294, 6.223],
       [1.324, 6.592],
       [1.1  , 4.548],
       [1.155, 5.432],
       [1.004, 6.3  ],
       [0.71 , 4.36 ],
       [0.385, 4.39 ],
       [0.913, 4.166],
       [1.124, 6.293],
       [1.286, 6.354],
       [0.446, 4.913],
       [0.948, 5.285],
       [1.221, 5.339],
       [0.85 , 4.559],
       [0.359, 3.334],
       [1.051, 5.523],
       [0.945, 5.386],
       [1.159, 6.444],
       [1.181, 5.287],
       [0.323, 3.597],
       [0.619, 3.462],
       [1.206, 6.182],
```

```
       [0.642, 5.86 ],
       [0.073, 3.975],
       [0.801, 5.208],
       [0.685, 5.529],
       [1.002, 5.211],
       [1.362, 6.199],
       [1.3  , 6.726],
       [1.376, 7.246],
       [1.057, 4.799],
       [1.183, 5.648],
       [0.38 , 4.534],
       [1.162, 6.07 ]])
```

In [430]:

```
# Fitting Optimized Multiple Linear Regression to the Training set
from sklearn.linear_model import LinearRegression
omr = LinearRegression().fit(X_train_2, y_train)
```

In [431]:

```
# Getting parameters
omr.coef_, omr.intercept_
```

Out[431]:

```
(array([0.55410166, 0.08626223, 0.39155437]), -0.011944142495915244)
```

In [432]:

```
# Predicting the Test set results
y_pred = omr.predict(X_test[:, 0:3])
```

In [433]:

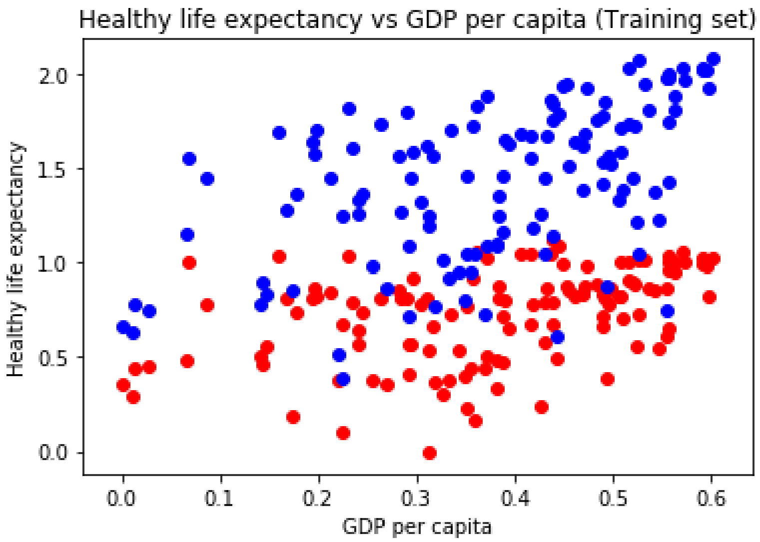
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, omr.predict(X_train[:, 0:3])), mean_squared_error(y_test, y_pr
ed)
```

Out[433]:

```
(0.5494366596653171, 0.5515984623853083)
```

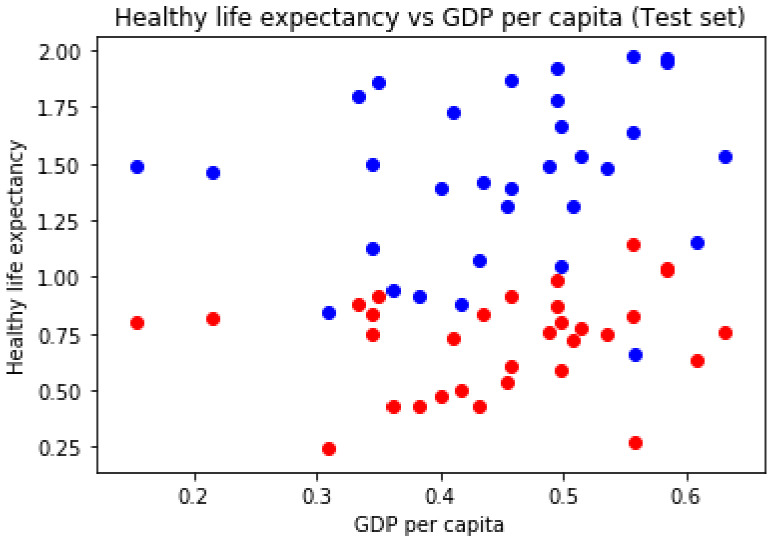
In [434]:

```
# Visualising the Training set results
plt.scatter(X_train[:,3], y_train, color = 'red')
plt.plot(X_train[:,3], omr.predict(X_train[:, 0:3]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [435]:

```
# Visualising the Test set results
plt.scatter(X_test[:,3], y_test, color = 'red')
plt.plot(X_test[:,3], omr.predict(X_test[:, 0:3]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



Regression Tree & Random Forest

In [436]:

```
df
```

Out[436]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
0	0.986	1.340	7.769	1.58700	0.596	0.393000	0.153
1	0.996	1.383	7.600	1.57300	0.592	0.394216	0.252
2	1.028	1.488	7.554	1.58200	0.603	0.341000	0.271
3	1.026	1.380	7.494	1.62400	0.591	0.118000	0.354
4	0.999	1.396	7.488	1.52200	0.557	0.298000	0.322
...	...	...	...	...	...	...	...
151	0.614	0.359	3.334	0.71100	0.555	0.394216	0.217
152	0.499	0.476	3.231	0.88500	0.417	0.147000	0.276
153	0.361	0.350	3.203	0.51700	0.000	0.025000	0.158
154	0.105	0.026	3.083	0.31124	0.225	0.035000	0.235
155	0.295	0.306	2.853	0.57500	0.010	0.091000	0.202

156 rows × 7 columns

In [437]:

```
# Splitting the dataset into the Training set and Test set
X = df.iloc[:, :-1].values
y = df.iloc[:, 6].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [438]:

```
# Fitting Tree to the Training set (Social support)
from sklearn.tree import DecisionTreeRegressor
sdt = DecisionTreeRegressor(max_leaf_nodes = 10).fit(X_train[:, 1:2], y_train)
```

In [439]:

```
# Predicting the Test set results
y_pred = sdt.predict(X_test[:, 1:2])
```

In [440]:

```
# Coefficient of determination R^2
sdt.score(X_train[:, 1:2], y_train), sdt.score(X_test[:, 1:2], y_test)
```

Out[440]:

```
(0.5447591487433597, -0.06954633380573516)
```

In [441]:

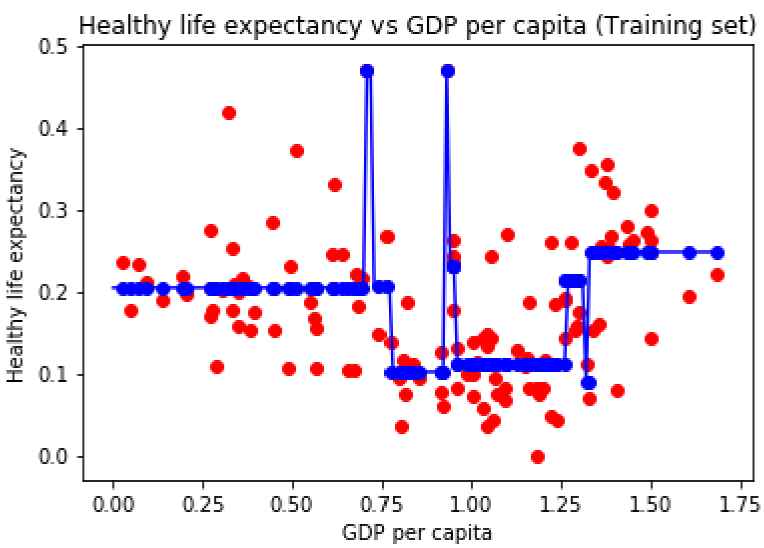
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, sdt.predict(X_train[:, 1:2])), mean_squared_error(y_test, y_pr
ed)
```

Out[441]:

```
(0.0037935883404230184, 0.009100451188306312)
```

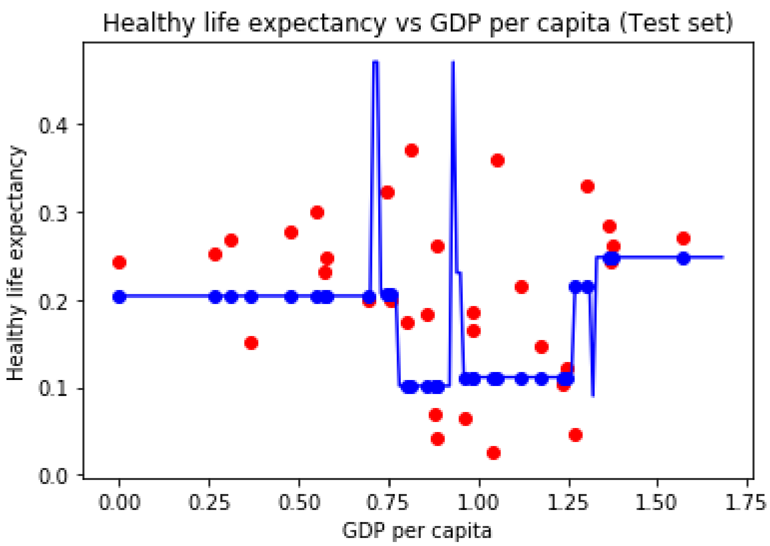
In [442]:

```
# Visualising the Training set results
X_grid = np.arange(min(X[:, 1:2]), max(X[:, 1:2]), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.plot(X_grid, sdt.predict(X_grid), color = 'blue')
plt.scatter(X_train[:,1], y_train, color = 'red')
plt.plot(X_train[:,1], sdt.predict(X_train[:, 1:2]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [443]:

```
# Visualising the Test set results
X_grid = np.arange(min(X[:, 1:2]), max(X[:, 1:2]), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.plot(X_grid, sdt.predict(X_grid), color = 'blue')
plt.scatter(X_test[:,1], y_test, color = 'red')
plt.plot(X_test[:,1], sdt.predict(X_test[:, 1:2]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [444]:

```
# Fitting Tree to the Training set
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor().fit(X_train[:, 0:1], y_train)
```

In [445]:

```
# Predicting the Test set results
y_pred = dt.predict(X_test[:, 0:1])
```

In [446]:

```
# Coefficient of determination R^2
dt.score(X_train[:, 0:1], y_train), dt.score(X_test[:, 0:1], y_test)
```

Out[446]:

(0.7328401898484327, -0.8029825954711218)

In [447]:

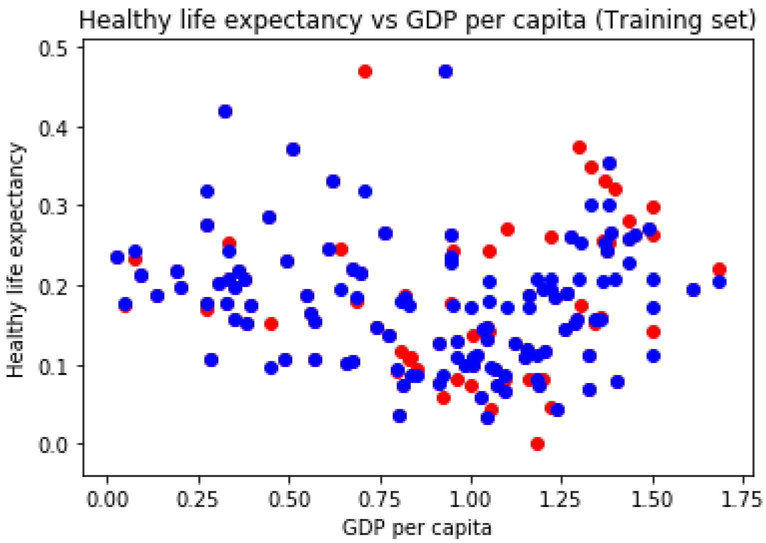
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, dt.predict(X_train[:, 0:1])), mean_squared_error(y_test, y_pre
d)
```

Out[447]:

```
(0.002226281622185219, 0.015341041883680557)
```

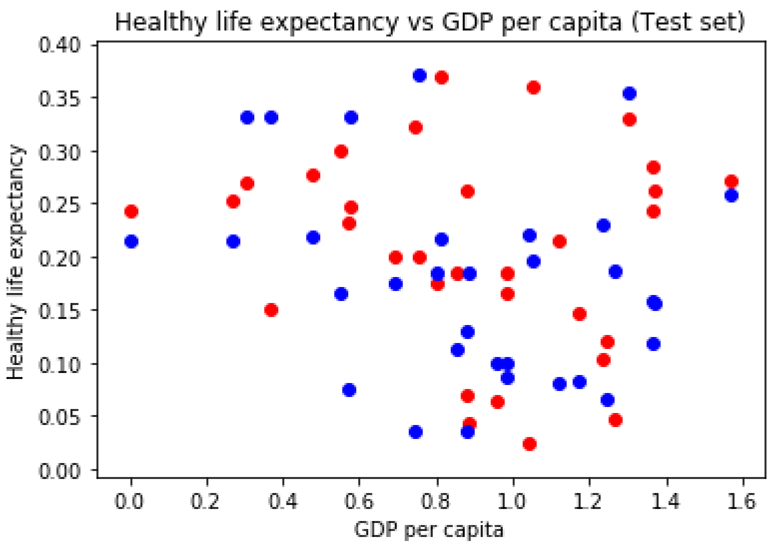
In [448]:

```
# Visualising the Training set results
plt.scatter(X_train[:,1], y_train, color = 'red')
plt.plot(X_train[:,1], dt.predict(X_train[:,0:1]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [449]:

```
# Visualising the Test set results
plt.scatter(X_test[:,1], y_test, color = 'red')
plt.plot(X_test[:,1], dt.predict(X_test[:, 0:1]), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [450]:

```
# Fitting Random Forest to the Training set
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators = 10, random_state = 0).fit(X_train, y_train)
```

In [451]:

```
# Predicting the Test set results
y_pred = rf.predict(X_test)
```

In [452]:

```
# Coefficient of determination R^2
rf.score(X_train, y_train), rf.score(X_test, y_test)
```

Out[452]:

```
(0.8467687185223698, 0.1764131181789248)
```

In [453]:

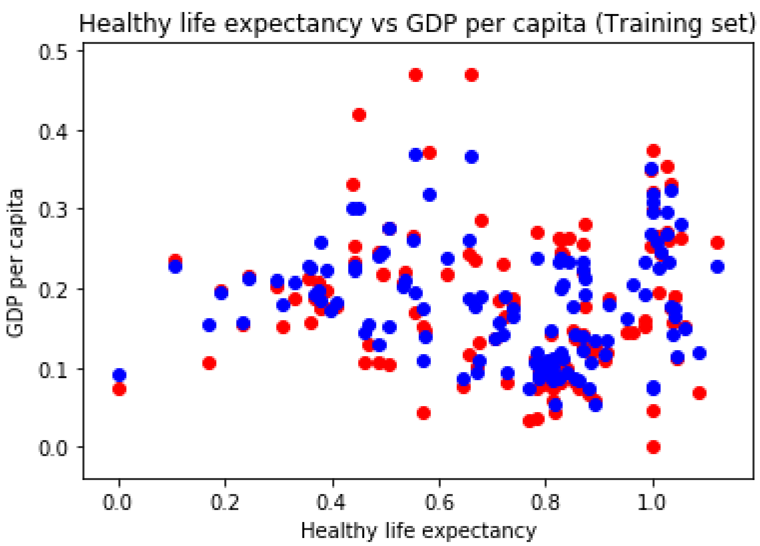
```
# Mean squared error
from sklearn.metrics import mean_squared_error
mean_squared_error(y_train, rf.predict(X_train)), mean_squared_error(y_test, y_pred)
```

Out[453]:

```
(0.001276898593781761, 0.007007655470775926)
```

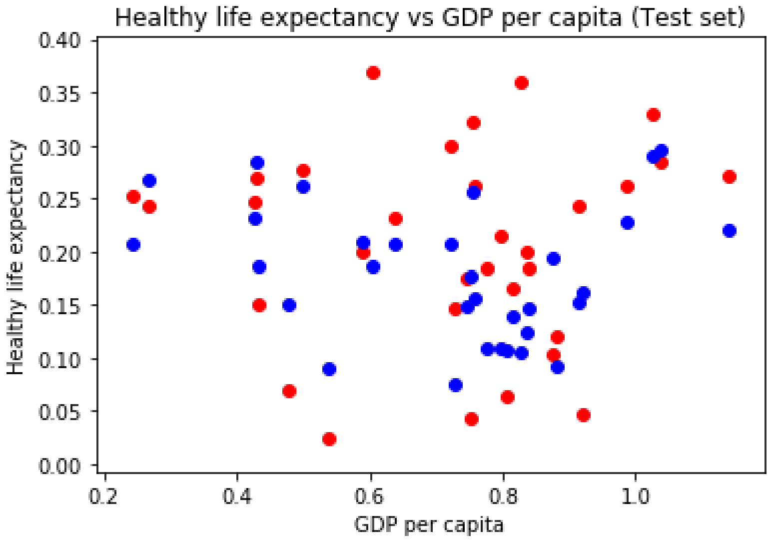
In [454]:

```
# Visualising the Training set results
plt.scatter(X_train[:,0], y_train, color = 'red')
plt.plot(X_train[:,0], rf.predict(X_train), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Training set)')
plt.xlabel('Healthy life expectancy')
plt.ylabel('GDP per capita')
plt.show()
```



In [455]:

```
# Visualising the Test set results
plt.scatter(X_test[:,0], y_test, color = 'red')
plt.plot(X_test[:,0], rf.predict(X_test), 'bo')
plt.title('Healthy life expectancy vs GDP per capita (Test set)')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



Regression Neural Network

In [456]:

```
df
```

Out[456]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
0	0.986	1.340	7.769	1.58700	0.596	0.393000	0.153
1	0.996	1.383	7.600	1.57300	0.592	0.394216	0.252
2	1.028	1.488	7.554	1.58200	0.603	0.341000	0.271
3	1.026	1.380	7.494	1.62400	0.591	0.118000	0.354
4	0.999	1.396	7.488	1.52200	0.557	0.298000	0.322
...	...	...	...	...	...	...	...
151	0.614	0.359	3.334	0.71100	0.555	0.394216	0.217
152	0.499	0.476	3.231	0.88500	0.417	0.147000	0.276
153	0.361	0.350	3.203	0.51700	0.000	0.025000	0.158
154	0.105	0.026	3.083	0.31124	0.225	0.035000	0.235
155	0.295	0.306	2.853	0.57500	0.010	0.091000	0.202

156 rows × 7 columns

In [457]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
dfsc = sc.fit_transform(df)
df['Score'] = dfsc[:,0]
df['GDP per capita'] = dfsc[:,1]
df['Social support'] = dfsc[:,2]
df['Healthy life expectancy'] = dfsc[:,3]
df['Freedom to make life choices'] = dfsc[:,4]
df['Generosity'] = dfsc[:,5]
df['Perceptions of corruption'] = dfsc[:,6]
```

In [458]:

```
# Cheking correlations
df.corr()
```

Out[458]:

	Healthy life expectancy	GDP per capita	Score	Social support	Freedom to make life choices	Perceptions of corruption	Generosity
Healthy life expectancy	1.000000	0.758243	0.719026	0.781755	0.450261	-0.046316	0.181722
GDP per capita	0.758243	1.000000	0.835462	0.793883	0.379079	-0.078898	0.298564
Score	0.719026	0.835462	1.000000	0.779883	0.390395	-0.025196	0.293698
Social support	0.781755	0.793883	0.779883	1.000000	0.566742	0.084708	0.390497
Freedom to make life choices	0.450261	0.379079	0.390395	0.566742	1.000000	0.270309	0.440441
Perceptions of corruption	-0.046316	-0.078898	-0.025196	0.084708	0.270309	1.000000	0.335468
Generosity	0.181722	0.298564	0.293698	0.390497	0.440441	0.335468	1.000000

In [459]:

```
# Splitting the dataset into the Training set and Test set
X = df.iloc[:, 1:6].values
y = df.iloc[:, 0].values
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [460]:

```
# Install Tensorflow
# Install Keras
# Importing the Keras libraries and packages
# !pip3 install keras
# !pip install tensorflow
import keras
from keras.models import Sequential
from keras.layers import Dense
```

In [461]:

```
# Initialising the ANN
rnn = Sequential()

# Adding the input layer and the first hidden layer
rnn.add(Dense(units = 6, activation = 'tanh', input_dim = 5))

# Adding the second hidden layer
rnn.add(Dense(units = 6, activation = 'tanh'))

# Adding the output layer
rnn.add(Dense(units = 1, activation = 'linear'))

# Compiling the ANN
rnn.compile(optimizer='adam', loss='mean_squared_error', metrics = ['accuracy'])
```

In [462]:

```
X_train = X_train.astype(np.float32)
```

In [463]:

```
# Fitting the ANN to the Training set
rnn.fit(X_train, y_train, batch_size = 10, epochs = 100)
```

```
Epoch 1/100
13/13 [==============================] - 0s 873us/step - loss: 0.6633 - accur
acy: 0.0000e+00
Epoch 2/100
13/13 [==============================] - 0s 1ms/step - loss: 0.5471 - accurac
y: 0.0000e+00
Epoch 3/100
13/13 [==============================] - 0s 921us/step - loss: 0.4681 - accur
acy: 0.0000e+00
Epoch 4/100
13/13 [==============================] - 0s 1ms/step - loss: 0.4260 - accurac
y: 0.0000e+00
Epoch 5/100
13/13 [==============================] - 0s 1ms/step - loss: 0.4069 - accurac
y: 0.0000e+00
Epoch 6/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3913 - accurac
y: 0.0000e+00
Epoch 7/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3848 - accurac
y: 0.0000e+00
Epoch 8/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3800 - accurac
y: 0.0000e+00
Epoch 9/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3778 - accurac
y: 0.0000e+00
Epoch 10/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3740 - accurac
y: 0.0000e+00
Epoch 11/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3751 - accurac
y: 0.0000e+00
Epoch 12/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3713 - accurac
y: 0.0000e+00
Epoch 13/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3696 - accurac
y: 0.0000e+00
Epoch 14/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3678 - accurac
y: 0.0000e+00
Epoch 15/100
13/13 [==============================] - 0s 967us/step - loss: 0.3681 - accur
acy: 0.0000e+00
Epoch 16/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3675 - accurac
y: 0.0000e+00
Epoch 17/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3627 - accurac
y: 0.0000e+00
Epoch 18/100
13/13 [==============================] - 0s 936us/step - loss: 0.3619 - accur
acy: 0.0000e+00
Epoch 19/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3610 - accurac
y: 0.0000e+00
```

```
Epoch 20/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3604 - accurac
y: 0.0000e+00
Epoch 21/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3594 - accurac
y: 0.0000e+00
Epoch 22/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3596 - accurac
y: 0.0000e+00
Epoch 23/100
13/13 [==============================] - 0s 944us/step - loss: 0.3561 - accur
acy: 0.0000e+00
Epoch 24/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3563 - accurac
y: 0.0000e+00
Epoch 25/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3548 - accurac
y: 0.0000e+00
Epoch 26/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3537 - accurac
y: 0.0000e+00
Epoch 27/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3562 - accurac
y: 0.0000e+00
Epoch 28/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3523 - accurac
y: 0.0000e+00
Epoch 29/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3514 - accurac
y: 0.0000e+00
Epoch 30/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3497 - accurac
y: 0.0000e+00
Epoch 31/100
13/13 [==============================] - 0s 997us/step - loss: 0.3495 - accur
acy: 0.0000e+00
Epoch 32/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3494 - accurac
y: 0.0000e+00
Epoch 33/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3481 - accurac
y: 0.0000e+00
Epoch 34/100
13/13 [==============================] - 0s 4ms/step - loss: 0.3473 - accurac
y: 0.0000e+00
Epoch 35/100
13/13 [==============================] - 0s 3ms/step - loss: 0.3472 - accurac
y: 0.0000e+00
Epoch 36/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3459 - accurac
y: 0.0000e+00
Epoch 37/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3448 - accurac
y: 0.0000e+00
Epoch 38/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3441 - accurac
y: 0.0000e+00
```

```
Epoch 39/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3437 - accurac
y: 0.0000e+00
Epoch 40/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3435 - accurac
y: 0.0000e+00
Epoch 41/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3433 - accurac
y: 0.0000e+00
Epoch 42/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3425 - accurac
y: 0.0000e+00
Epoch 43/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3413 - accurac
y: 0.0000e+00
Epoch 44/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3409 - accurac
y: 0.0000e+00
Epoch 45/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3406 - accurac
y: 0.0000e+00
Epoch 46/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3403 - accurac
y: 0.0000e+00
Epoch 47/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3395 - accurac
y: 0.0000e+00
Epoch 48/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3398 - accurac
y: 0.0000e+00
Epoch 49/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3380 - accurac
y: 0.0000e+00
Epoch 50/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3375 - accurac
y: 0.0000e+00
Epoch 51/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3377 - accurac
y: 0.0000e+00
Epoch 52/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3367 - accurac
y: 0.0000e+00
Epoch 53/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3367 - accurac
y: 0.0000e+00
Epoch 54/100
13/13 [==============================] - 0s 993us/step - loss: 0.3358 - accur
acy: 0.0000e+00
Epoch 55/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3344 - accurac
y: 0.0000e+00
Epoch 56/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3348 - accurac
y: 0.0000e+00
Epoch 57/100
13/13 [==============================] - 0s 865us/step - loss: 0.3348 - accur
acy: 0.0000e+00
```

```
Epoch 58/100
13/13 [==============================] - 0s 844us/step - loss: 0.3340 - accur
acy: 0.0000e+00
Epoch 59/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3339 - accurac
y: 0.0000e+00
Epoch 60/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3320 - accurac
y: 0.0000e+00
Epoch 61/100
13/13 [==============================] - 0s 3ms/step - loss: 0.3328 - accurac
y: 0.0000e+00
Epoch 62/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3333 - accurac
y: 0.0000e+00
Epoch 63/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3313 - accurac
y: 0.0000e+00
Epoch 64/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3317 - accurac
y: 0.0000e+00
Epoch 65/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3321 - accurac
y: 0.0000e+00
Epoch 66/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3313 - accurac
y: 0.0000e+00
Epoch 67/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3289 - accurac
y: 0.0000e+00
Epoch 68/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3297 - accurac
y: 0.0000e+00
Epoch 69/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3286 - accurac
y: 0.0000e+00
Epoch 70/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3281 - accurac
y: 0.0000e+00
Epoch 71/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3279 - accurac
y: 0.0000e+00
Epoch 72/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3271 - accurac
y: 0.0000e+00
Epoch 73/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3266 - accurac
y: 0.0000e+00
Epoch 74/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3275 - accurac
y: 0.0000e+00
Epoch 75/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3263 - accurac
y: 0.0000e+00
Epoch 76/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3251 - accurac
y: 0.0000e+00
```

```
Epoch 77/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3259 - accurac
y: 0.0000e+00
Epoch 78/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3239 - accurac
y: 0.0000e+00
Epoch 79/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3257 - accurac
y: 0.0000e+00
Epoch 80/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3238 - accurac
y: 0.0000e+00
Epoch 81/100
13/13 [==============================] - ETA: 0s - loss: 0.6208 - accuracy:
0.0000e+ - 0s 3ms/step - loss: 0.3232 - accuracy: 0.0000e+00
Epoch 82/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3241 - accurac
y: 0.0000e+00
Epoch 83/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3217 - accurac
y: 0.0000e+00
Epoch 84/100
13/13 [==============================] - 0s 3ms/step - loss: 0.3213 - accurac
y: 0.0000e+00
Epoch 85/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3223 - accurac
y: 0.0000e+00
Epoch 86/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3218 - accurac
y: 0.0000e+00
Epoch 87/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3206 - accurac
y: 0.0000e+00
Epoch 88/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3205 - accurac
y: 0.0000e+00
Epoch 89/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3201 - accurac
y: 0.0000e+00
Epoch 90/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3199 - accurac
y: 0.0000e+00
Epoch 91/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3195 - accurac
y: 0.0000e+00
Epoch 92/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3188 - accurac
y: 0.0000e+00
Epoch 93/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3196 - accurac
y: 0.0000e+00
Epoch 94/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3172 - accurac
y: 0.0000e+00
Epoch 95/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3167 - accurac
y: 0.0000e+00
```

```
Epoch 96/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3166 - accurac
y: 0.0000e+00
Epoch 97/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3164 - accurac
y: 0.0000e+00
Epoch 98/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3154 - accurac
y: 0.0000e+00
Epoch 99/100
13/13 [==============================] - 0s 1ms/step - loss: 0.3148 - accurac
y: 0.0000e+00
Epoch 100/100
13/13 [==============================] - 0s 2ms/step - loss: 0.3141 - accurac
y: 0.0000e+00
```

Out[463]:

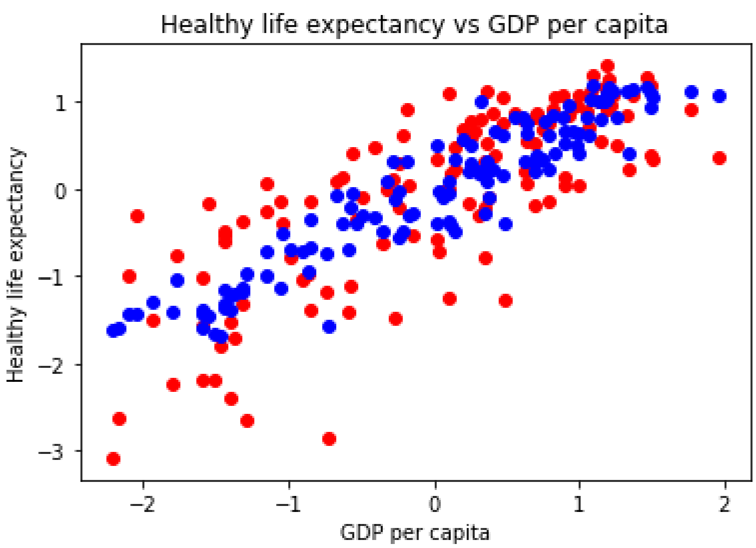
```
<tensorflow.python.keras.callbacks.History at 0x1553041c8b0>
```

In [464]:

```
# Predicting the Test set results
y_pred = rnn.predict(X_test)
```

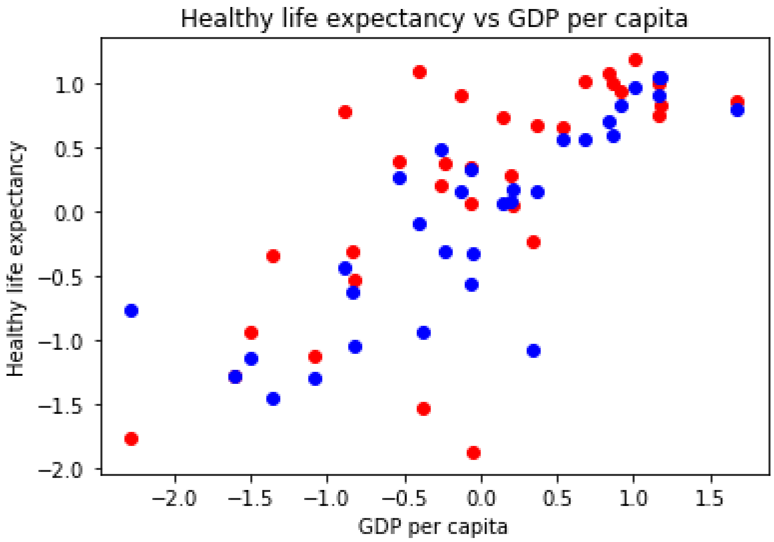
In [465]:

```
# Visualising the Training set results
plt.scatter(X_train[:,0], y_train, color = 'red')
plt.plot(X_train[:,0], rnn.predict(X_train), 'bo')
plt.title('Healthy life expectancy vs GDP per capita')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [466]:

```
# Visualising the Test set results
plt.scatter(X_test[:,0], y_test, color = 'red')
plt.plot(X_test[:,0], rnn.predict(X_test), 'bo')
plt.title('Healthy life expectancy vs GDP per capita')
plt.xlabel('GDP per capita')
plt.ylabel('Healthy life expectancy')
plt.show()
```



In [467]:

```
# Mean Squared Error
y_pred_train = rnn.predict(X_train)
y_pred_test = rnn.predict(X_test)
train_mse_nn = sum((y_train - y_pred_train) ** 2 for y_train, y_pred_train in zip(y_train,
y_pred_train)) / len(y_train)
test_mse_nn = sum((y_test - y_pred_test) ** 2 for y_test, y_pred_test in zip(y_test, y_pre
d_test)) / len(y_test)
print(f"train_mse_nn: {train_mse_nn}, test_mse_nn: {test_mse_nn}")
```

```
train_mse_nn: [0.31337714], test_mse_nn: [0.39204016]
```

In [468]:

```
# Как вывод наблюдаем, что по метрике MSE для тестовой виборки лучшей является модель Mult
iple Linear Regression.
# Также отметим, что достаточно неплохой оказалась модель Random Forest, Polynomial Regres
sion, Simple Linear Regression и Regression Tree

# Модели Regression Neural Network и Backward Elimination with p-values совершенно не подх
одят для данного датасета.

# Таким образом, лучшими моделями по критерию MSE для тестовой выборки являются Regression
Tree, Random Forest и Regression Neural Network.

# Ниже расположены краткие оценки моделей по убыванию.

# Multiple Linear Regression
# Coefficient of determination R^2
# (0.7151031441860636, 0.8468652866187845)
# Mean squared error
# (0.017416369138027004, 0.007205518852440761)

# Random Forest
# Coefficient of determination R^2
# (0.846603322934345, 0.7474906564201986)
# Mean squared error
# (0.001265387519124442, 0.015341041883680557)

# Polynomial Regression
# Coefficient of determination R^2
# (0.6832933964115014, 0.7725186782820886)
# Mean squared error
# (0.019360968729505563, 0.010703784374063528)

# Simple Linear Regression
# Coefficient of determination R^2
# (0.6007918214228924, 0.6447212155182563)
# Mean squared error
# (0.02440447080805655, 0.01671709779534196)

# Regression Tree
# Coefficient of determination R^2
# (0.5401220284055326, 0.6954633380573516)
# Mean squared error
# (0.0037935883404230184, 0.009100451188306312)

# Regression Neural Network
# Mean squared error
# (0.3195072, 0.3098919)

# Backward Elimination with p-values
# Mean squared error
# (0.5494366596653171, 0.5515984623853083)
```