In [1]:
```
# Packages for data manipulation
import numpy as np
import pandas as pd
import warnings

# Packages for plotting
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

# to run SQL code
import sqlite3
import IPython

# Packages for data modeling
from scipy import stats
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix
import statsmodels.stats.outliers_influence as inf
from sklearn.naive_bayes import MultinomialNB
import statsmodels.tools.tools as stattools

# Packages for decision trees
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn import tree

# Packages for splitting data into test and train sets
from sklearn.model_selection import train_test_split
import random
```

In [2]:
```
# Second row of file contains information on data types, used "skiprows=[1]" to skip this row on import
df = pd.read_csv( '/Users/oscargil/Downloads/factbook.csv', sep=',', skiprows=[1])
```

In [3]:
```
# Initial shape of imported data, 263 columns, 45 rows
df.shape
```

Out[3]:
```
(263, 45)
```

In [4]:
```
# Clean up column names, stripping them of extra spaces
df.columns = (df.columns.str.strip().str.replace('-', '').str.replace(' ','_').str.replace('__','_'))
```

In [5]:
```
# declare SQL database connection
cnn = sqlite3.connect('df.db')
```

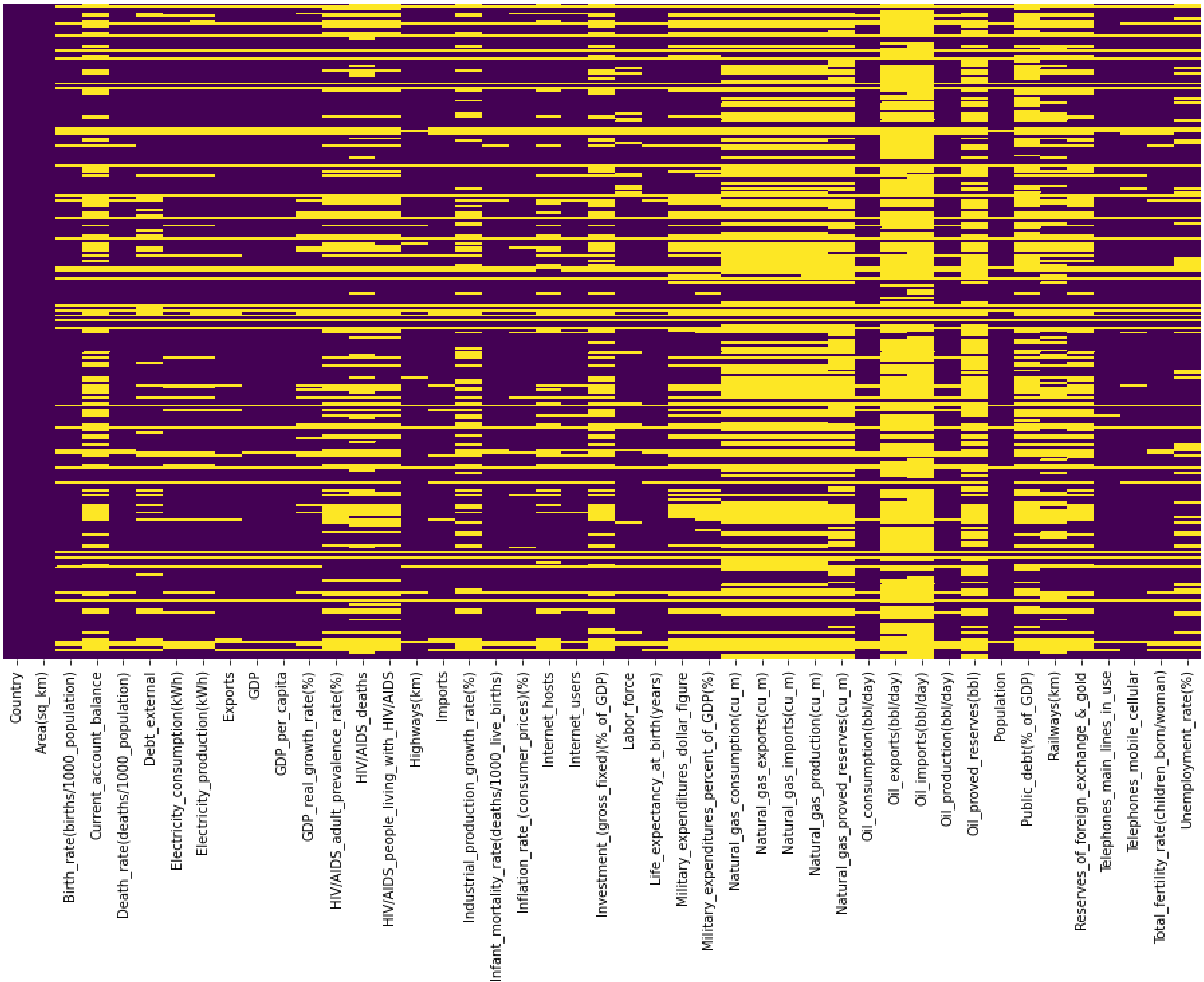
In [6]:
```
# load dataframe into database
df.to_sql("df", cnn, if_exists='replace')
```

In [7]:
```
# load sql extension and connect to database
%load_ext sql
%sql sqlite:///df.db
```

Out[7]:
```
'Connected: @df.db'
```

In [8]:
```
# view dataset's null data, represented by yellow lines
plt.figure(figsize=(16,9))
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

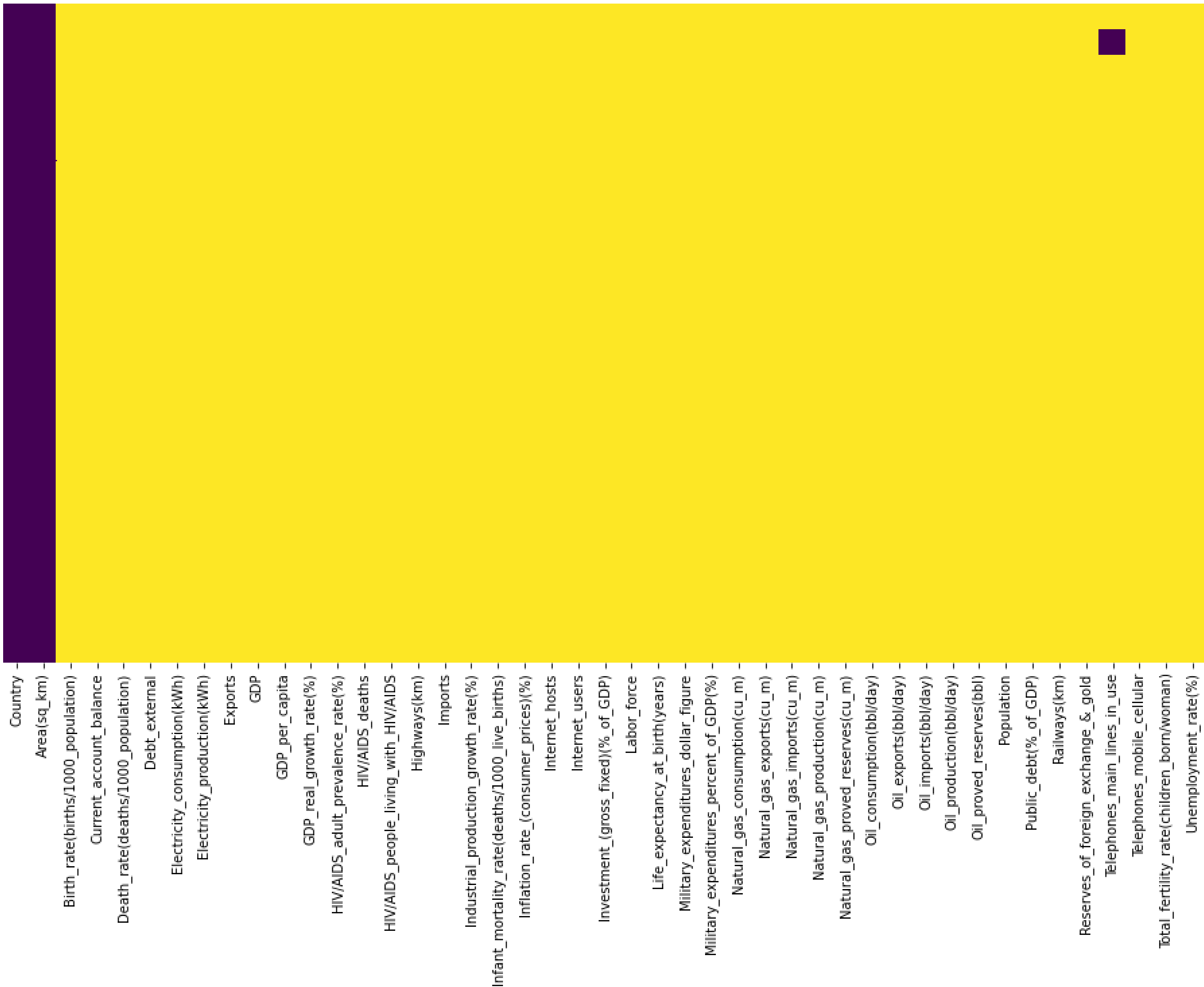
Out[8]:
```
<AxesSubplot:>
```



In [9]:

```
# Plot all data related to null Population
null_population = df[df['Population'].isnull()]

plt.figure(figsize=(16,9))
sns.heatmap(null_population.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

Out[9]: <AxesSubplot:>



In [10]:

```
# drop all rows where population is null, justified by the data produced in plot above
df.dropna(subset=['Population'],inplace=True)
```

In [11]:

```
# drop specific columns missing too much data
df.drop('Natural_gas_consumption(cu_m)',axis=1,inplace=True)
df.drop('Natural_gas_exports(cu_m)',axis=1,inplace=True)
df.drop('Natural_gas_imports(cu_m)',axis=1,inplace=True)
df.drop('Natural_gas_production(cu_m)',axis=1,inplace=True)
df.drop('Natural_gas_proved_reserves(cu_m)',axis=1,inplace=True)
df.drop('Oil_exports(bbl/day)',axis=1,inplace=True)
df.drop('Oil_imports(bbl/day)',axis=1,inplace=True)
df.drop('Oil_proved_reserves(bbl)',axis=1,inplace=True)
df.drop('Public_debt(%_of_GDP)',axis=1,inplace=True)
df.drop('Current_account_balance',axis=1,inplace=True)
df.drop('Investment_(gross_fixed)(%_of_GDP)',axis=1,inplace=True)
df.drop('Railways(km)',axis=1,inplace=True)
df.drop('Reserves_of_foreign_exchange_&_gold',axis=1,inplace=True)
df.drop('Military_expenditures_dollar_figure',axis=1,inplace=True)
df.drop('Military_expenditures_percent_of_GDP(%)',axis=1,inplace=True)
df.drop('Industrial_production_growth_rate(%)',axis=1,inplace=True)
df.drop('HIV/AIDS_adult_prevalence_rate(%)',axis=1,inplace=True)
df.drop('HIV/AIDS_deaths',axis=1,inplace=True)
df.drop('HIV/AIDS_people_living_with_HIV/AIDS',axis=1,inplace=True)
```

In [12]:

```
# Shape of data after dropping rows associated to Population where all rows were null
# and dropping columns missing too much data
# We now have 238 rows and 26 columns
df.shape
```

Out[12]: (238, 26)

In [13]:

```
# remaining column names with total null value rows
null_columns = df.columns[df.isnull().any()]
nulls = df[null_columns].isnull().sum()
```

```
# load dataframe into database
nulls.to_sql("nulls", cnn, if_exists='replace')

%sql select [index] as 'Column', [0] as TotalNulls
   , printf("%.2f",cast([0] as float) / 238) as NullPct
    from nulls order by 3 desc
```

```
 * sqlite:///df.db
Done.
```

Out[13]:

Column	TotalNulls	NullPct
Internet_hosts	47	0.20
Unemployment_rate(%)	46	0.19
Debt_external	37	0.16
Electricity_production(kWh)	25	0.11
GDP_real_growth_rate(%)	26	0.11
Oil_consumption(bbl/day)	26	0.11
Oil_production(bbl/day)	26	0.11
Electricity_consumption(kWh)	23	0.10
Labor_force	24	0.10
Internet_users	22	0.09
Inflation_rate_(consumer_prices)(%)	16	0.07
Exports	14	0.06
Imports	14	0.06
Birth_rate(births/1000_population)	13	0.05
Death_rate(deaths/1000_population)	13	0.05
Infant_mortality_rate(deaths/1000_live_births)	13	0.05
Life_expectancy_at_birth(years)	13	0.05
Total_fertility_rate(children_born/woman)	13	0.05
Telephones_mobile_cellular	10	0.04
GDP	8	0.03
GDP_per_capita	8	0.03
Highways(km)	8	0.03
Telephones_main_lines_in_use	7	0.03

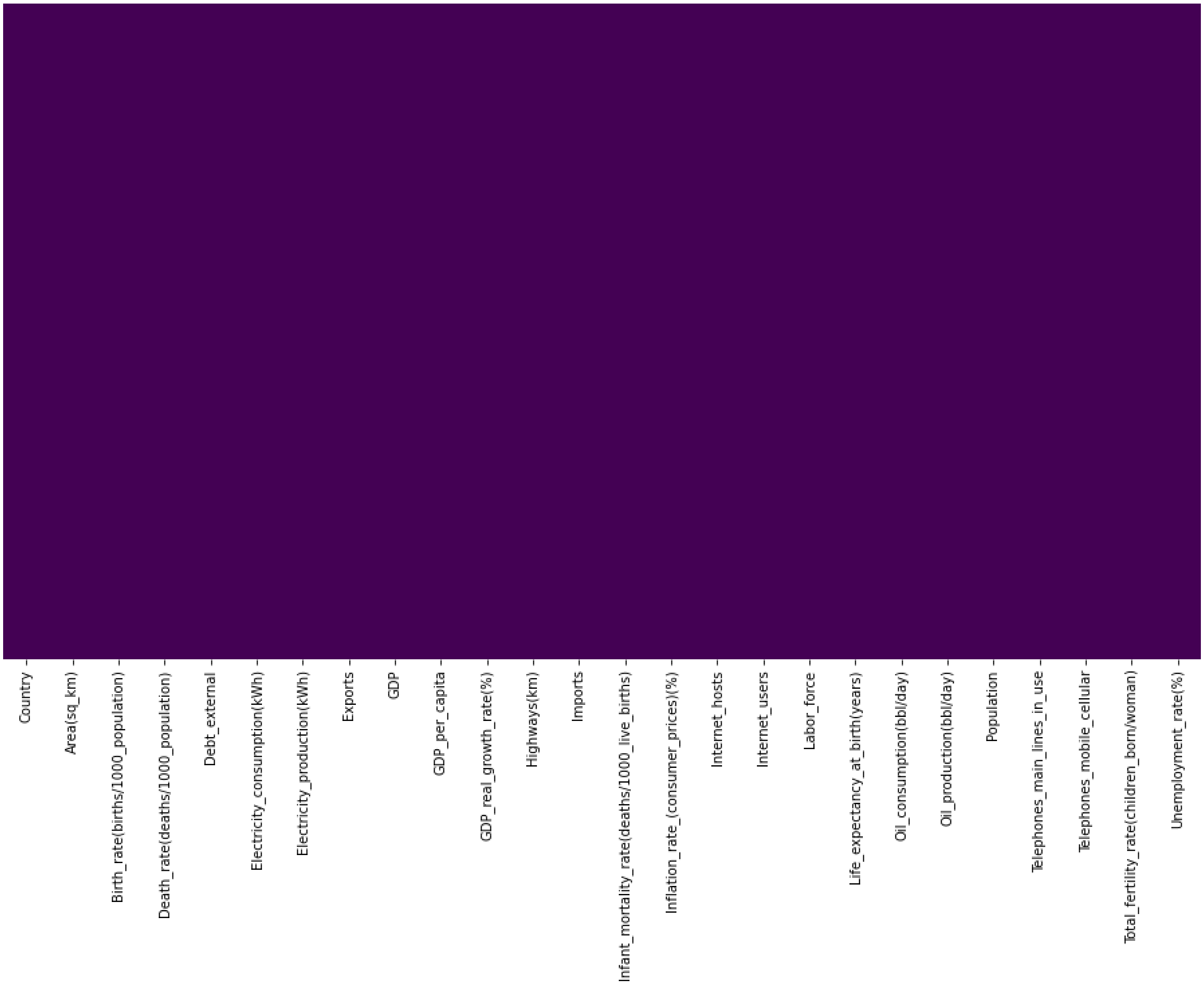
In [14]:

```
# set remaining null values to the mean of the column
df['Birth_rate(births/1000_population)'].fillna((df['Birth_rate(births/1000_population)'].mean()),inplace=True)
df['Death_rate(deaths/1000_population)'].fillna((df['Death_rate(deaths/1000_population)'].mean()),inplace=True)
df['Debt_external'].fillna((df['Debt_external'].mean()),inplace=True)
df['Electricity_consumption(kWh)'].fillna((df['Electricity_consumption(kWh)'].mean()),inplace=True)
df['Electricity_production(kWh)'].fillna((df['Electricity_production(kWh)'].mean()),inplace=True)
df['Exports'].fillna((df['Exports'].mean()),inplace=True)
df['GDP'].fillna((df['GDP'].mean()),inplace=True)
df['GDP_per_capita'].fillna((df['GDP_per_capita'].mean()),inplace=True)
df['GDP_real_growth_rate(%)'].fillna((df['GDP_real_growth_rate(%)'].mean()),inplace=True)
df['Highways(km)'].fillna((df['Highways(km)'].mean()),inplace=True)
df['Imports'].fillna((df['Imports'].mean()),inplace=True)
df['Infant_mortality_rate(deaths/1000_live_births)'].fillna((df['Infant_mortality_rate(deaths/1000_live_births)'].mean()),in
df['Inflation_rate_(consumer_prices)(%)'].fillna((df['Inflation_rate_(consumer_prices)(%)'].mean()),inplace=True)
df['Internet_hosts'].fillna((df['Internet_hosts'].mean()),inplace=True)
df['Internet_users'].fillna((df['Internet_users'].mean()),inplace=True)
df['Labor_force'].fillna((df['Labor_force'].mean()),inplace=True)
df['Life_expectancy_at_birth(years)'].fillna((df['Life_expectancy_at_birth(years)'].mean()),inplace=True)
df['Oil_consumption(bbl/day)'].fillna((df['Oil_consumption(bbl/day)'].mean()),inplace=True)
df['Oil_production(bbl/day)'].fillna((df['Oil_production(bbl/day)'].mean()),inplace=True)
df['Telephones_main_lines_in_use'].fillna((df['Telephones_main_lines_in_use'].mean()),inplace=True)
df['Telephones_mobile_cellular'].fillna((df['Telephones_mobile_cellular'].mean()),inplace=True)
df['Total_fertility_rate(children_born/woman)'].fillna((df['Total_fertility_rate(children_born/woman)'].mean()),inplace=True
df['Unemployment_rate(%)'].fillna((df['Unemployment_rate(%)'].mean()),inplace=True)
```

In [15]:

```
plt.figure(figsize = (16,9))
sns.heatmap(df.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

Out[15]: <AxesSubplot:>

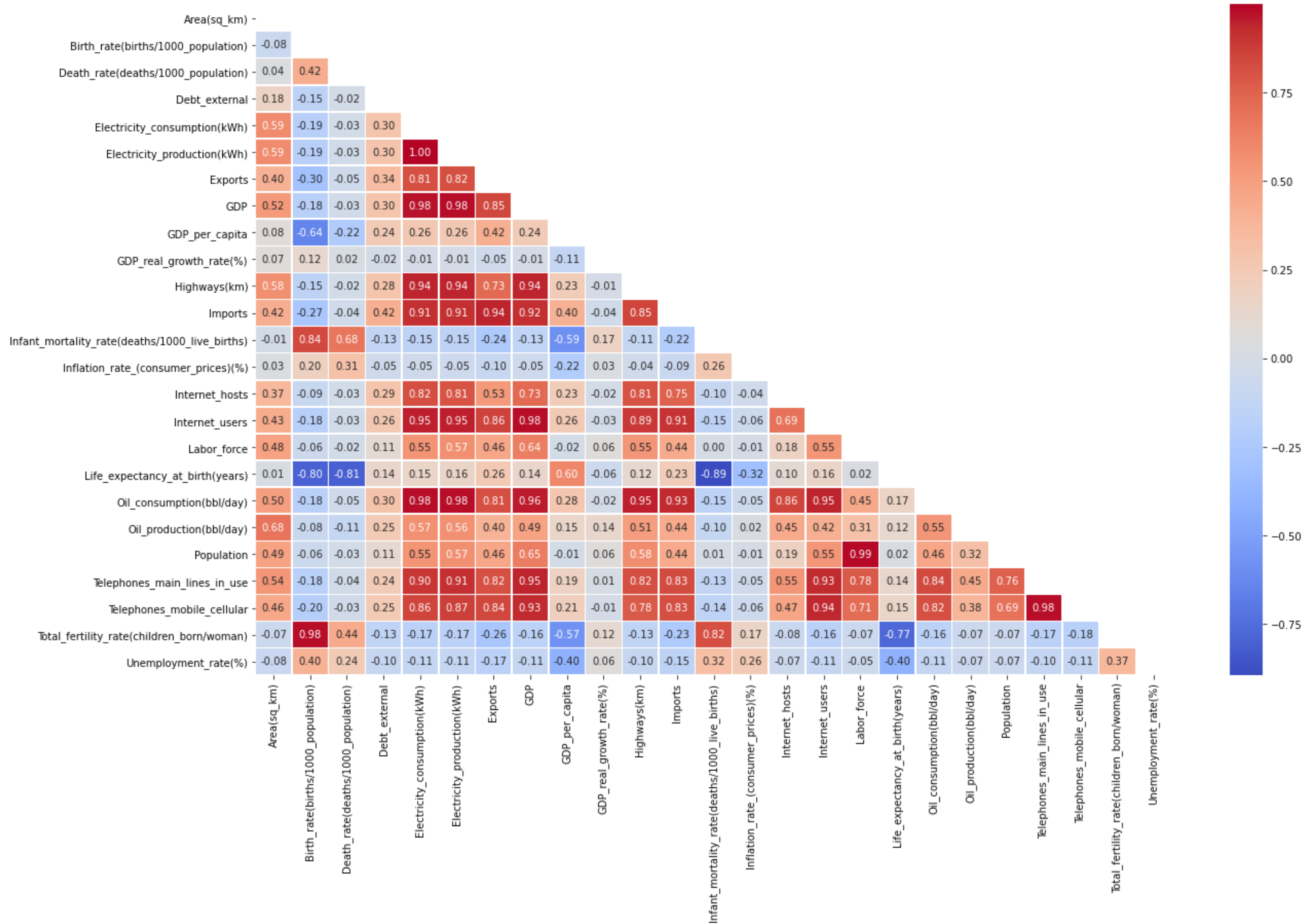


In [16]:

```
# Pearson correlation matrix
pearsoncorr = df.corr(method='pearson')
plt.figure(figsize = (20,12))
sns.heatmap(pearsoncorr,
mask= np.triu(df.corr()),
xticklabels=pearsoncorr.columns, yticklabels=pearsoncorr.columns, cmap="coolwarm",
fmt=".2f",
annot=True, linewidth=0.5)
```

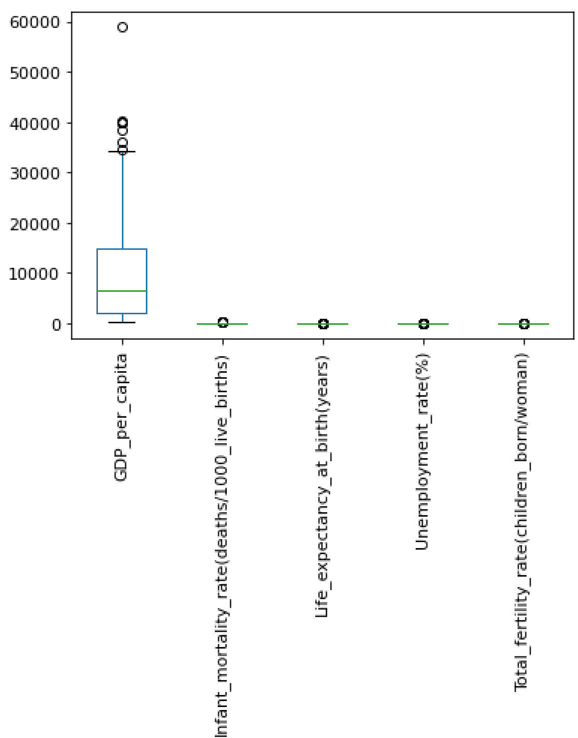
Out[16]:

```
<AxesSubplot:>
```



In [17]:

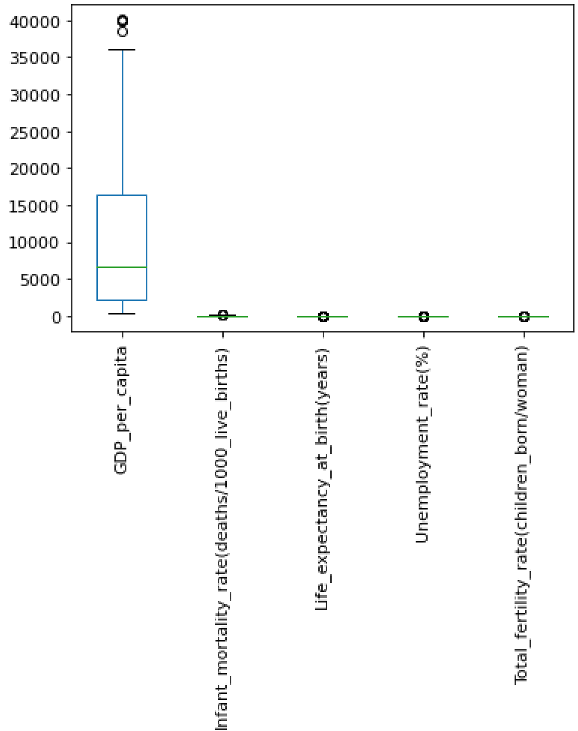
```
df2 = pd.DataFrame(df[['GDP_per_capita',
                       'Infant_mortality_rate(deaths/1000_live_births)',
                       'Life_expectancy_at_birth(years)',
                       'Unemployment_rate(%)',
                       'Total_fertility_rate(children_born/woman)']])
boxplot = df2.boxplot(grid=False, rot=90, fontsize=11)
```



In [18]:

```
#This removes our outliers with a z-score of greater than 3
df2 = df2[(np.abs(stats.zscore(df2)) < 3).all(axis=1)]
```

```
boxplot = df2.boxplot(grid=False, rot=90, fontsize=11)
```



In [19]:

```
df2['GDP_per_capita'].describe()
```

Out[19]:

```
count      229.000000
mean     10589.975223
std      10557.426261
min        400.000000
25%       2200.000000
50%       6600.000000
75%      16400.000000
max      40100.000000
Name: GDP_per_capita, dtype: float64
```

In [20]:

```
df2['GDP_per_capita_bin'] = np.where(df2['GDP_per_capita'].between(0,6600),0,1)
```

In [21]:

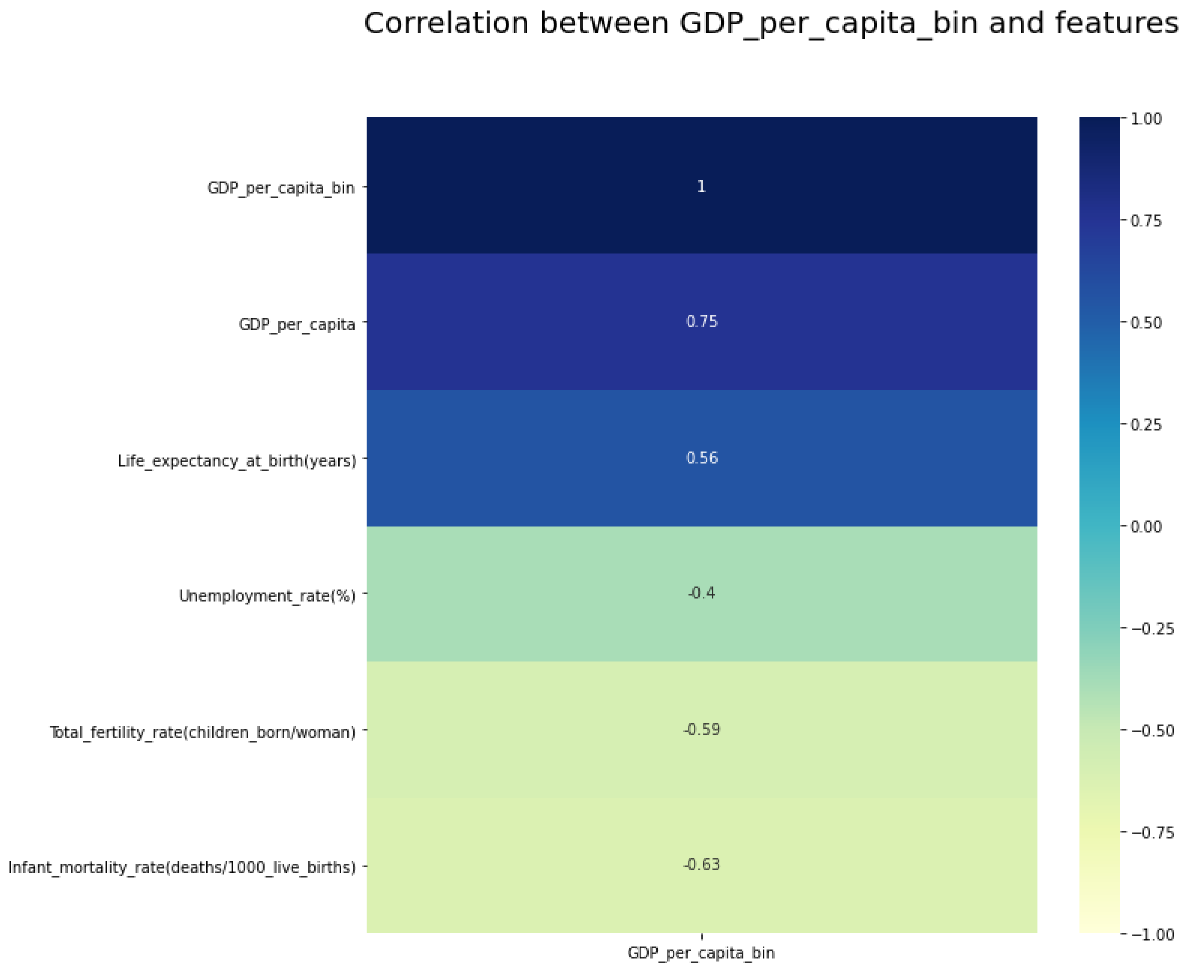
```
df2['GDP_per_capita_bin'].value_counts()
```

Out[21]:

```
0    116
1    113
Name: GDP_per_capita_bin, dtype: int64
```

In [22]:

```
fig, ax = plt.subplots(figsize=(10,10))
fig.suptitle('Correlation between GDP_per_capita_bin and features',fontsize=20)
ax=sns.heatmap(df2.corr()[["GDP_per_capita_bin"]].sort_values("GDP_per_capita_bin"),vmax=1, vmin=-1, cmap="YlGnBu"
               , annot=True, ax=ax);
ax.invert_yaxis()
```

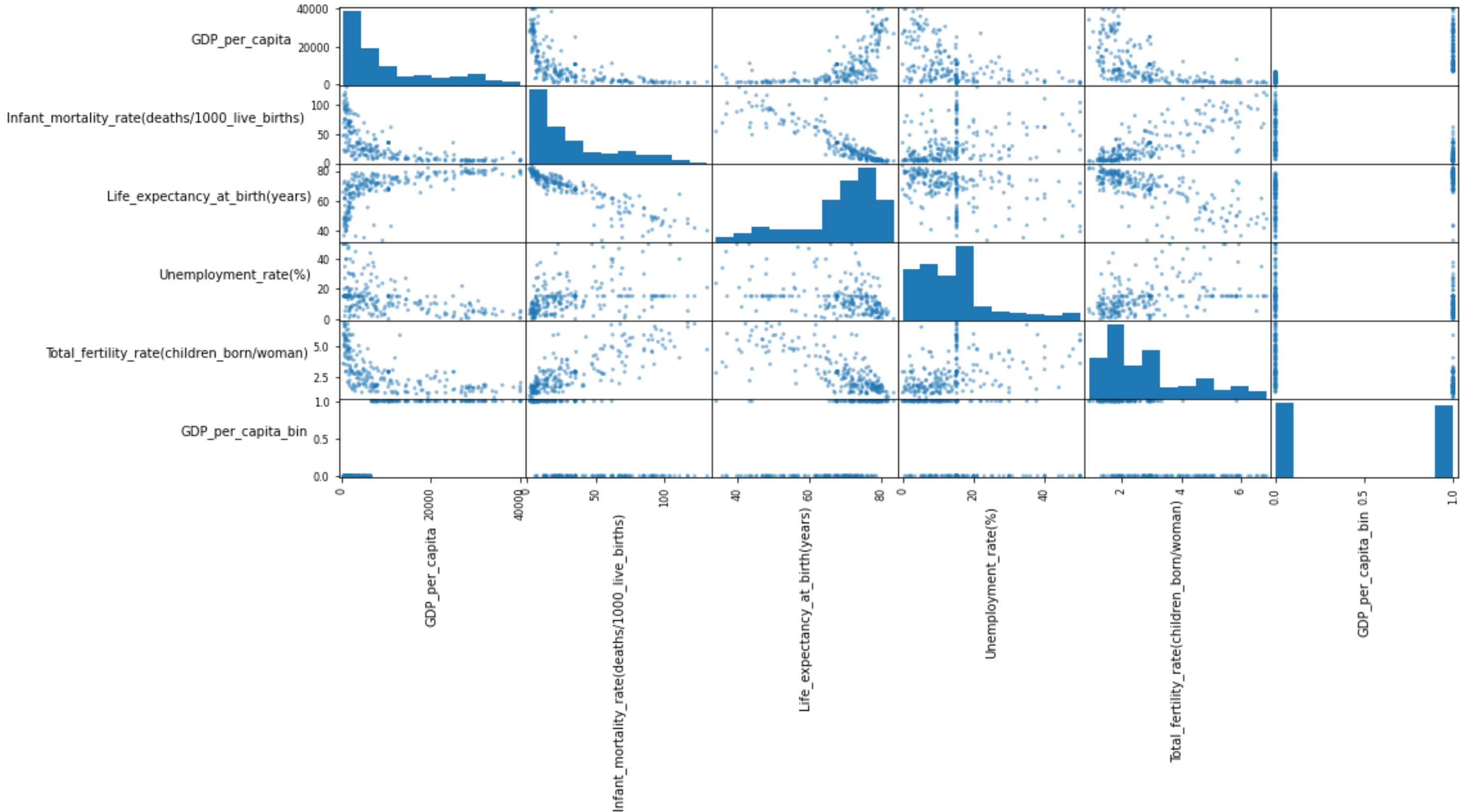


Here we can see that the correlation values seem to be very good.

In [23]:

```
plt.figure(figsize = (16,9))
axes = pd.plotting.scatter_matrix(df2, figsize=(16,9))
for ax in axes.flatten():
    ax.xaxis.label.set_rotation(90)
    ax.yaxis.label.set_rotation(0)
    ax.yaxis.label.set_ha('right')

plt.tight_layout()
plt.gcf().subplots_adjust(wspace=0, hspace=0)
plt.show()
```

```
<Figure size 1152x648 with 0 Axes>
```



Looking at the above figure, we can see there are no multicolinearity issues, allowing us to begin testing classification models.

In [24]:

```
#Splitting the data, leaving 25% for testing
df2_train, df2_test = train_test_split(df2, test_size = 0.25, random_state = 7)
```

In [25]:

```
print(" Original = ", len(df2), '\n',
"Test Size = ", len(df2_test), '\n',
"Train Size = ", len(df2_train))
```

```
Original =  229
Test Size =  58
Train Size =  171
```

In [26]:

```
list(df2_train)
```

Out[26]:

```
['GDP_per_capita',
 'Infant_mortality_rate(deaths/1000_live_births)',
 'Life_expectancy_at_birth(years)',
 'Unemployment_rate(%)',
 'Total_fertility_rate(children_born/woman)',
 'GDP_per_capita_bin']
```

In [27]:

```
X = pd.DataFrame(df2_train[[ 'Infant_mortality_rate(deaths/1000_live_births)',
                            'Life_expectancy_at_birth(years)',
                            'Unemployment_rate(%)',
                            'Total_fertility_rate(children_born/woman)']])
y = pd.DataFrame(df2_train[['GDP_per_capita_bin']])
```

In [28]:

```
X.head()
```

Out[28]:

	Infant_mortality_rate(deaths/1000_live_births)	Life_expectancy_at_birth(years)	Unemployment_rate(%)	Total_fertility_rate(children_born/woman)
40	69.29	43.50	15.254688	5.81
201	19.00	77.76	14.000000	1.54
245	67.83	51.59	15.254688	6.74
41	71.48	58.87	2.500000	3.44
18	17.27	74.23	15.000000	2.63

In [29]:

```
X = X.rename(columns={'Infant_mortality_rate(deaths/1000_live_births)'  : 'Infant_mortality_rate',
                      'Total_fertility_rate(children_born/woman)'       : 'Total_fertility_rate',
                      'Life_expectancy_at_birth(years)'                 : 'Life_expectancy_at_birth'} )
```

In [30]:
```
X.head()
```

Out[30]:

	Infant_mortality_rate	Life_expectancy_at_birth	Unemployment_rate(%)	Total_fertility_rate
40	69.29	43.50	15.254688	5.81
201	19.00	77.76	14.000000	1.54
245	67.83	51.59	15.254688	6.74
41	71.48	58.87	2.500000	3.44
18	17.27	74.23	15.000000	2.63

In [31]:
```
df2_train['GDP_per_capita_bin'].value_counts()
```

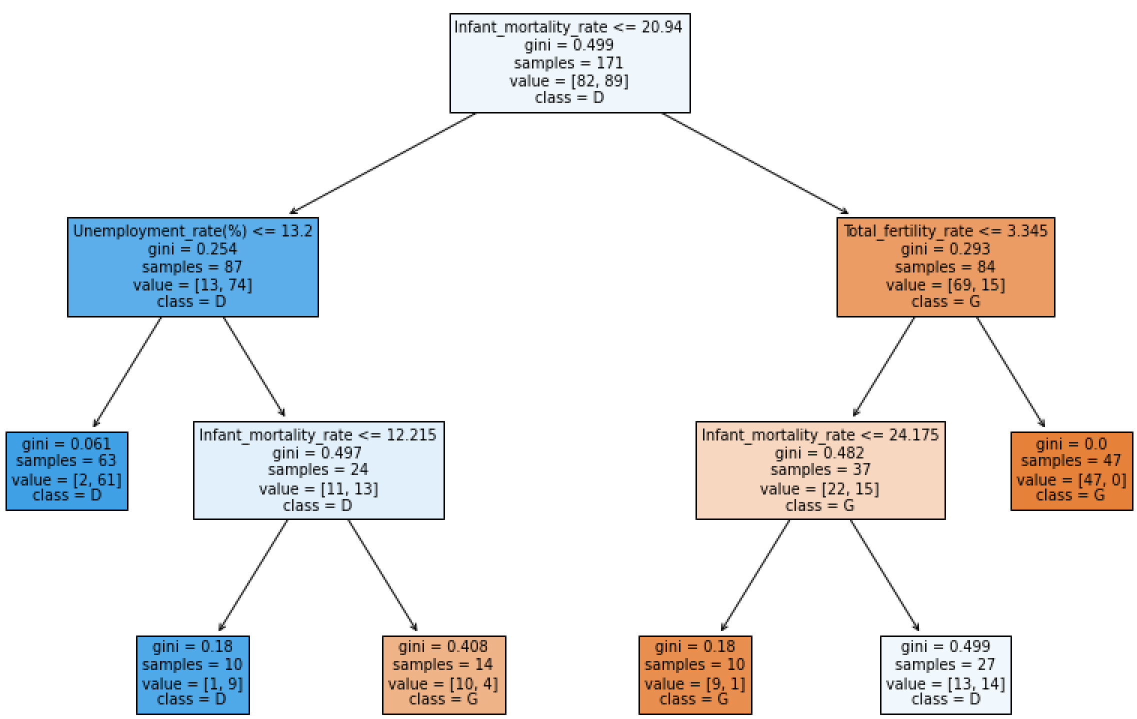
Out[31]:
```
1    89
0    82
Name: GDP_per_capita_bin, dtype: int64
```

In [32]:
```
df2_test['GDP_per_capita_bin'].value_counts()
```

Out[32]:
```
0    34
1    24
Name: GDP_per_capita_bin, dtype: int64
```

Building a CART Decision Tree

In [33]:
```
cart01 = DecisionTreeClassifier(criterion="gini", max_leaf_nodes= 6).fit(X,y)
fig = plt.figure(figsize=(15,10))

tree.plot_tree(cart01,
               feature_names=list(X),
               class_names="GDP_per_capita_bin",
               filled=True)

plt.show()
```



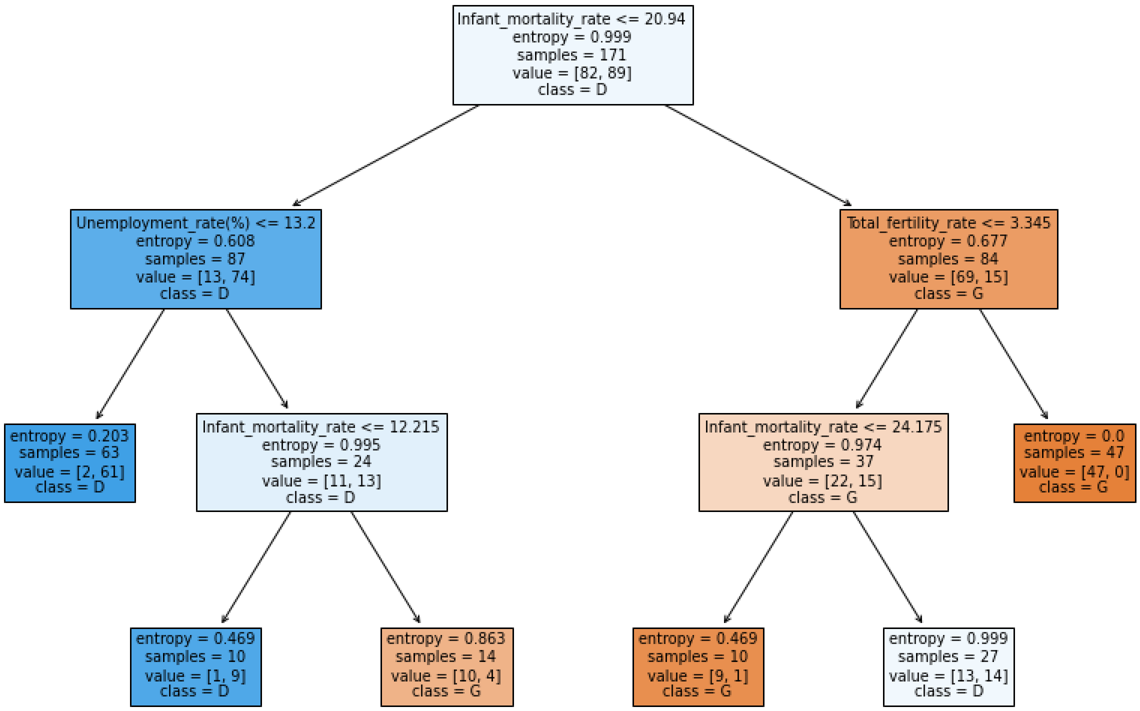
In [34]:
```
predGDP = cart01.predict(X)
table = confusion_matrix(y, predGDP)
print(table)
```

```
[[66 16]
 [ 5 84]]
```

Building a C5.0 model

In [35]:
```
c50_01 = DecisionTreeClassifier(criterion="entropy", max_leaf_nodes=6).fit(X,y)
```

In [36]:
```
fig = plt.figure(figsize=(15,10))
_ = tree.plot_tree(c50_01,
                   feature_names=list(X),
                   class_names="GDP_per_capita_bin",
                   filled=True)
```



In [37]:
```
predGDP2 = c50_01.predict(X)
```

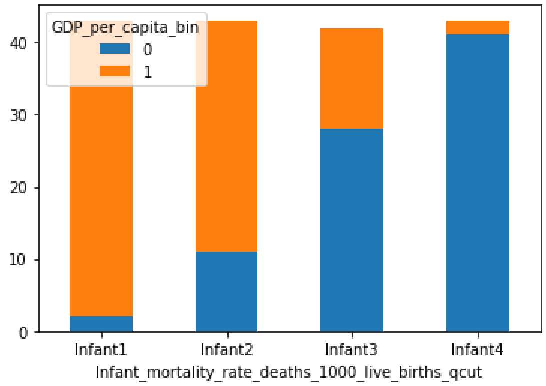
In [38]:
```
table2 = confusion_matrix(y, predGDP2)
print(table2)
```

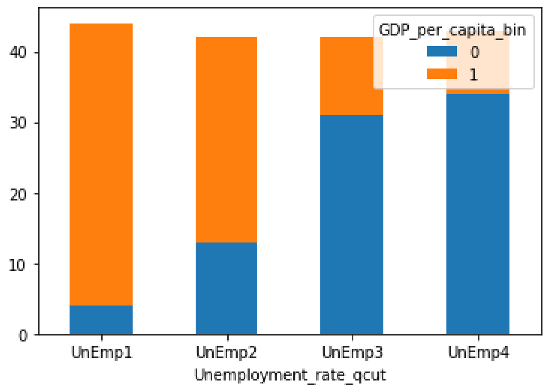
```
[[66 16]
 [ 5 84]]
```

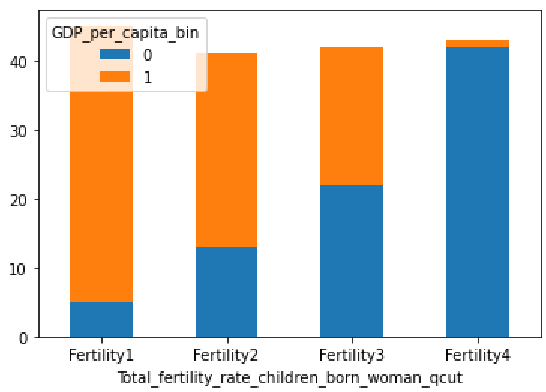
Building a Naive Bayes model

In [39]:
```
warnings.filterwarnings('ignore')

# Plot probabilities...
# Variables were selected from CART and C5.0 results

df2_train['Infant_mortality_rate_deaths_1000_live_births_qcut'] = pd.qcut( df2_train['Infant_mortality_rate(deaths/1000_live_
t1 = pd.crosstab(df2_train['Infant_mortality_rate_deaths_1000_live_births_qcut'], df2_train['GDP_per_capita_bin'])
t1.plot(kind = 'bar', stacked = True, rot = 0)


df2_train['Unemployment_rate_qcut'] = pd.qcut( df2_train['Unemployment_rate(%)'], q = 4, labels = ['UnEmp1','UnEmp2','UnEmp3
t2 = pd.crosstab(df2_train['Unemployment_rate_qcut'], df2_train['GDP_per_capita_bin'])
t2.plot(kind = 'bar', stacked = True, rot = 0)


df2_train['Total_fertility_rate_children_born_woman_qcut'] = pd.qcut( df2_train['Total_fertility_rate(children_born/woman)']
t3 = pd.crosstab(df2_train['Total_fertility_rate_children_born_woman_qcut'], df2_train['GDP_per_capita_bin'])
t3.plot(kind = 'bar', stacked = True, rot = 0)
```

Out[39]: `<AxesSubplot:xlabel='Total_fertility_rate_children_born_woman_qcut'>`







In [40]:

```
# ...convert "train" variables
X_Infant_ind    = pd.get_dummies(df2_train['Infant_mortality_rate_deaths_1000_live_births_qcut'])
X_UnEmp_ind     = pd.get_dummies(df2_train['Unemployment_rate_qcut'])
X_Fertility_ind = pd.get_dummies(df2_train['Total_fertility_rate_children_born_woman_qcut'])


#...convert "test" variables
df2_test['Infant_mortality_rate_deaths_1000_live_births_qcut'] = pd.qcut( df2_test['Infant_mortality_rate(deaths/1000_live_bi
df2_test['Unemployment_rate_qcut'] = pd.qcut( df2_test['Unemployment_rate(%)'], q = 4, labels = ['UnEmp1','UnEmp2','UnEmp3',
df2_test['Total_fertility_rate_children_born_woman_qcut'] = pd.qcut( df2_test['Total_fertility_rate(children_born/woman)'], q

X_Infant_ind_test    = pd.get_dummies(df2_test['Infant_mortality_rate_deaths_1000_live_births_qcut'])
X_UnEmp_ind_test     = pd.get_dummies(df2_test['Unemployment_rate_qcut'])
X_Fertility_ind_test = pd.get_dummies(df2_test['Total_fertility_rate_children_born_woman_qcut'])
```

In [41]:

```
# Naïve Bayes
Y      = df2_train['GDP_per_capita_bin']
Y_test = df2_test['GDP_per_capita_bin']
```

In [42]:

```
nb_01 = MultinomialNB().fit(X_Infant_ind, Y)
nb01score = nb_01.score(X_Infant_ind, Y)

nb_02 = MultinomialNB().fit(X_UnEmp_ind, Y)
nb02score = nb_02.score(X_UnEmp_ind, Y)

nb_03 = MultinomialNB().fit(X_Fertility_ind, Y)
nb03score = nb_03.score(X_Fertility_ind, Y)

print(" Infant_mortality_rate(deaths/1000_live_births) SCORE = ", nb01score, '\n',
      "Unemployment_rate SCORE                             = ", nb02score, '\n',
      "Total_fertility_rate_children_born_woman SCORE      = ", nb03score)
```

```
 Infant_mortality_rate(deaths/1000_live_births) SCORE =  0.8304093567251462
```

```
Unemployment_rate SCORE                                    =  0.783625730994152
Total_fertility_rate_children_born_woman SCORE             =  0.7719298245614035
```

In [43]:
```
#...generate predictions
Y_predicted_01 = nb_01.predict(X_Infant_ind_test)

Y_predicted_02 = nb_02.predict(X_UnEmp_ind_test)

Y_predicted_03 = nb_03.predict(X_Fertility_ind_test)
```

In [44]:
```
#... accuracy scores
from sklearn.metrics import accuracy_score

acc01 = accuracy_score(Y_test, Y_predicted_01)
acc02 = accuracy_score(Y_test, Y_predicted_02)
acc03 = accuracy_score(Y_test, Y_predicted_03)

print(" Infant_mortality_rate(deaths/1000_live_births) ACCURACY = ", acc01, '\n',
      "Unemployment_rate ACCURACY                             = ", acc02, '\n',
      "Total_fertility_rate_children_born_woman ACCURACY       = ", acc03)
```

```
Infant_mortality_rate(deaths/1000_live_births) ACCURACY =  0.7413793103448276
Unemployment_rate ACCURACY                              =  0.5689655172413793
Total_fertility_rate_children_born_woman ACCURACY       =  0.7068965517241379
```

In [45]:
```
# Contingency tables
ypred01 = pd.crosstab(Y_test, Y_predicted_01, rownames = ['Actual (Infant_mortality_rate)'], colnames = ['Predicted'])
ypred01['Total'] = ypred01.sum(axis=1); ypred01.loc['Total'] = ypred01.sum();
ypred01
```

Out[45]:

Predicted	0	1	Total
Actual (Infant_mortality_rate)			
0	24	10	34
1	5	19	24
Total	29	29	58

In [46]:
```
ypred02 = pd.crosstab(Y_test, Y_predicted_02, rownames = ['Actual (Unemployment_rate)'], colnames = ['Predicted'])
ypred02['Total'] = ypred02.sum(axis=1); ypred02.loc['Total'] = ypred02.sum();
ypred02
```

Out[46]:

Predicted	0	1	Total
Actual (Unemployment_rate)			
0	19	15	34
1	10	14	24
Total	29	29	58

In [47]:
```
ypred03 = pd.crosstab(Y_test, Y_predicted_03, rownames = ['Actual (Total_fertility_rate)'], colnames = ['Predicted'])
ypred03['Total'] = ypred03.sum(axis=1); ypred03.loc['Total'] = ypred03.sum();
ypred03
```

Out[47]:

Predicted	0	1	Total
Actual (Total_fertility_rate)			
0	23	11	34
1	6	18	24
Total	29	29	58

Logistic Regression

In [48]:
```
# First attempt...
# Variable 'Total_fertility_rate(children_born/woman)' has a high p-value
# ...going to re-run, removing this variable
X_1 = pd.DataFrame(df2_train[['Infant_mortality_rate(deaths/1000_live_births)','Unemployment_rate(%)','Total_fertility_rate(c
X_1 = sm.add_constant(X_1)
y_1 = pd.DataFrame(df2_train['GDP_per_capita_bin'])

logreg01 = sm.Logit(y_1, X_1).fit()

logreg01.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.361846
         Iterations 8
```

Out[48]:

Model:	Logit	Pseudo R-squared:	0.477
Dependent Variable:	GDP_per_capita_bin	AIC:	131.7514
Date:	2021-08-16 13:09	BIC:	144.3181
No. Observations:	171	Log-Likelihood:	-61.876
Df Model:	3	LL-Null:	-118.38
Df Residuals:	167	LLR p-value:	2.4587e-24
Converged:	1.0000	Scale:	1.0000
No. Iterations:	8.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	4.4178	0.7684	5.7492	0.0000	2.9117	5.9239
Infant_mortality_rate(deaths/1000_live_births)	-0.0634	0.0188	-3.3798	0.0007	-0.1001	-0.0266
Unemployment_rate(%)	-0.0857	0.0278	-3.0791	0.0021	-0.1402	-0.0311
Total_fertility_rate(children_born/woman)	-0.5746	0.3083	-1.8639	0.0623	-1.1789	0.0296

In [49]:

```
# Second attempt...
# Looks good, the independent variables are within acceptable p-value
X_2 = pd.DataFrame(df2_train[['Infant_mortality_rate(deaths/1000_live_births)','Unemployment_rate(%)']])
X_2 = sm.add_constant(X_2)
y_2 = pd.DataFrame(df2_train['GDP_per_capita_bin'])

logreg02 = sm.Logit(y_2, X_2).fit()

logreg02.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.373185
         Iterations 8
```

Out[49]:

Model:	Logit	Pseudo R-squared:	0.461
Dependent Variable:	GDP_per_capita_bin	AIC:	133.6293
Date:	2021-08-16 13:09	BIC:	143.0543
No. Observations:	171	Log-Likelihood:	-63.815
Df Model:	2	LL-Null:	-118.38
Df Residuals:	168	LLR p-value:	1.9974e-24
Converged:	1.0000	Scale:	1.0000
No. Iterations:	8.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	3.5609	0.5663	6.2875	0.0000	2.4508	4.6709
Infant_mortality_rate(deaths/1000_live_births)	-0.0828	0.0161	-5.1392	0.0000	-0.1144	-0.0512
Unemployment_rate(%)	-0.0930	0.0276	-3.3701	0.0008	-0.1471	-0.0389

In [50]:

```
# Model validation using test data
# The 'Unemployment_rate(%)' variable came back with a high p-value; however, the R-squared value
# almost remained the same, dropping by 0.059.
X_3 = pd.DataFrame(df2_test[['Infant_mortality_rate(deaths/1000_live_births)','Unemployment_rate(%)']])
X_3 = sm.add_constant(X_3)
y_3 = pd.DataFrame(df2_test['GDP_per_capita_bin'])

logreg03 = sm.Logit(y_3, X_3).fit()

logreg03.summary2()
```

```
Optimization terminated successfully.
         Current function value: 0.405572
         Iterations 7
```

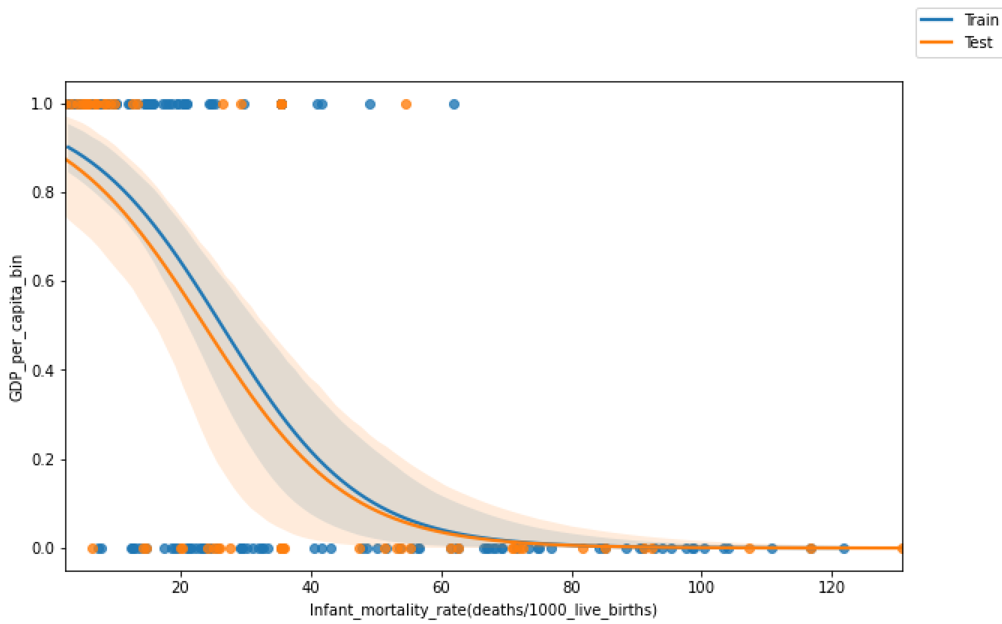
Out[50]:

Model:	Logit	Pseudo R-squared:	0.402
Dependent Variable:	GDP_per_capita_bin	AIC:	53.0464
Date:	2021-08-16 13:09	BIC:	59.2277
No. Observations:	58	Log-Likelihood:	-23.523
Df Model:	2	LL-Null:	-39.336
Df Residuals:	55	LLR p-value:	1.3568e-07

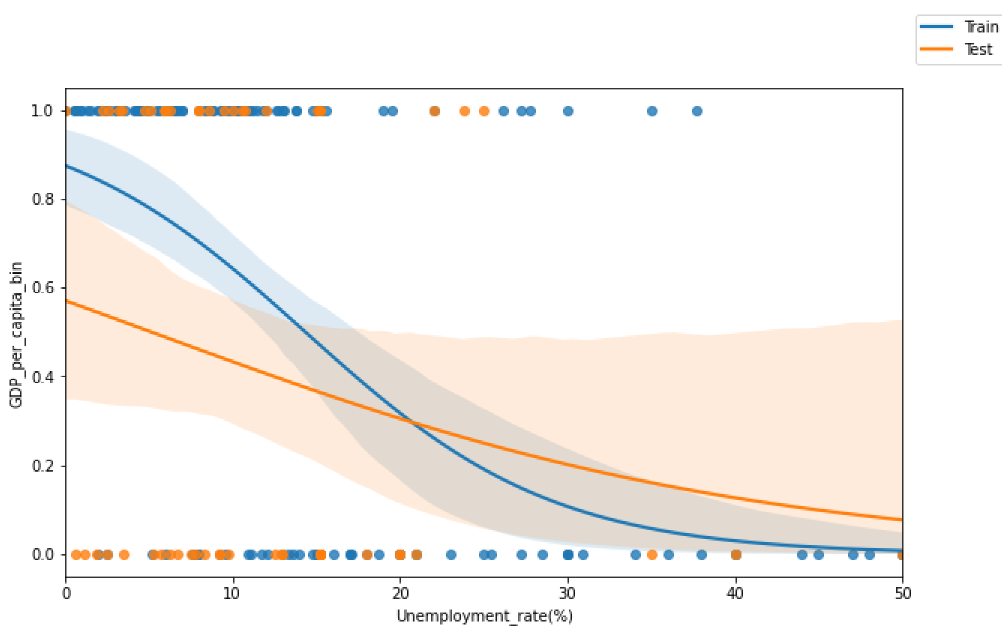
Converged:	1.0000	Scale:	1.0000
No. Iterations:	7.0000		

	Coef.	Std.Err.	z	P>|z|	[0.025	0.975]
const	2.0587	0.7196	2.8608	0.0042	0.6483	3.4691
Infant_mortality_rate(deaths/1000_live_births)	-0.0935	0.0271	-3.4451	0.0006	-0.1467	-0.0403
Unemployment_rate(%)	0.0140	0.0486	0.2871	0.7741	-0.0814	0.1093

In [51]:
```
# Plot of train and test data for 'Infant_mortality_rate(deaths/1000_live_births)' against 'GDP_per_capita_bin'
fig = plt.figure(figsize=(10,6))
sns.regplot(x='Infant_mortality_rate(deaths/1000_live_births)', y='GDP_per_capita_bin', data=df2_train, logistic=True)
sns.regplot(x='Infant_mortality_rate(deaths/1000_live_births)', y='GDP_per_capita_bin', data=df2_test, logistic=True)
fig.legend(labels=['Train','Test'])
plt.show()
```



In [52]:
```
# Plot of train and test data for 'Unemployment_rate(%)' against 'GDP_per_capita_bin'
fig = plt.figure(figsize=(10,6))
sns.regplot(x='Unemployment_rate(%)', y='GDP_per_capita_bin', data=df2_train, logistic=True)
sns.regplot(x='Unemployment_rate(%)', y='GDP_per_capita_bin', data=df2_test, logistic=True)
fig.legend(labels=['Train','Test'])
plt.show()
```



In [53]:
```
# Plot of train and test data for 'Total_fertility_rate(children_born/woman)' against 'GDP_per_capita_bin'
fig = plt.figure(figsize=(10,6))
sns.regplot(x='Total_fertility_rate(children_born/woman)', y='GDP_per_capita_bin', data=df2_train, logistic=True)
sns.regplot(x='Total_fertility_rate(children_born/woman)', y='GDP_per_capita_bin', data=df2_test, logistic=True)
```

```
fig.legend(labels=['Train','Test'])
plt.show()
```