In [1]:

```
# to handle datasets
import pandas as pd
import numpy as np

# for plotting
import matplotlib.pyplot as plt
%matplotlib inline

# to divide train and test set
from sklearn.model_selection import train_test_split

# feature scaling
from sklearn.preprocessing import StandardScaler

# for tree binarisation
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score


# to build the models
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
import xgboost as xgb

# to evaluate the models
from sklearn.metrics import mean_squared_error

pd.pandas.set_option('display.max_columns', None)

import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# load dataset
df = pd.read_csv('dat102_cleaned.csv')
print(df.shape)
df.head()
```

```
(1280, 47)
```

Out[2]:

	row_id	country_code	year	agricultural_land_area	percentage_of_arable_land_equ
0	1248	1881055	2000	12364.30652	27.873476
1	887	1881055	2001	12292.06398	27.627966
2	969	1881055	2002	12402.83412	27.724205
3	993	1881055	2003	12244.51139	28.454720
4	591	1881055	2004	12171.79164	28.514809

In [3]:
```
# Load the dataset for submission (the one on which our model will b
e evaluated by Kaggle)
# it contains exactly the same variables, but not the target

submission = pd.read_csv('dat102testcleaned.csv')
submission.head()
```

Out[3]:

	row_id	country_code	year	agricultural_land_area	percentage_of_arable_land_equ
0	0	c3ce4bf	2012	15020.598000	99.417846
1	1	1af00b8	2009	11899.695840	54.339839
2	2	dbc74a3	2015	573171.401200	33.070081
3	3	c72199a	2003	157313.946300	4.793346
4	4	8191231	2001	340.049453	33.070081

In [4]:
```
# find categorical variables
categorical = [var for var in df.columns if df[var].dtype=='O']
print('There are {} categorical variables'.format(len(categorical)))
```

```
There are 1 categorical variables
```

In [5]:
```
# find numerical variables
numerical = [var for var in df.columns if df[var].dtype!='O']
print('There are {} numerical variables'.format(len(numerical)))
```

```
There are 46 numerical variables
```

In [6]:
```
#explore any missing years for train set
pd.set_option('display.max_rows',100)
table1 = pd.crosstab(df['country_code'],df['year'])
display(table1)
```

year	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
country_code												
04952a0	1	1	1	1	1	1	1	1	1	1	1	1
0593aa0	1	1	1	1	1	1	1	1	1	1	1	1
066b021	1	1	1	1	1	1	1	1	1	1	1	1
07f8d11	1	1	1	1	1	1	1	1	1	1	1	1
085807f	1	1	1	1	1	1	1	1	1	1	1	1
0b6e276	1	1	1	1	1	1	1	1	1	1	1	1
0c0177b	1	1	1	1	1	1	1	1	1	1	1	1
0ea781c	1	1	1	1	1	1	1	1	1	1	1	1
100c476	1	1	1	1	1	1	1	1	1	1	1	1
11c9833	1	1	1	1	1	1	1	1	1	1	1	1
12c8f8f	1	1	1	1	1	1	1	1	1	1	1	1
1881055	1	1	1	1	1	1	1	1	1	1	1	1
22b9653	1	1	1	1	1	1	1	1	1	1	1	1
2ca26c6	1	1	1	1	1	1	1	1	1	1	1	1
2ddc563	1	1	1	1	1	1	1	1	1	1	1	1
2e5e810	1	1	1	1	1	1	1	1	1	1	1	1
2f1d47e	1	1	1	1	1	1	1	1	1	1	1	1
30e2302	1	1	1	1	1	1	1	1	1	1	1	1
3e049d7	1	1	1	1	1	1	1	1	1	1	1	1
4080343	1	1	1	1	1	1	1	1	1	1	1	1
42c298b	1	1	1	1	1	1	1	1	1	1	1	1
45a15a2	1	1	1	1	1	1	1	1	1	1	1	1
4609682	1	1	1	1	1	1	1	1	1	1	1	1
508731a	1	1	1	1	1	1	1	1	1	1	1	1
583201c	1	1	1	1	1	1	1	1	1	1	1	1
5c2e474	1	1	1	1	1	1	1	1	1	1	1	1
5dbddf9	1	1	1	1	1	1	1	1	1	1	1	1
5f1162c	1	1	1	1	1	1	1	1	1	1	1	1
6.30E+87	1	1	1	1	1	1	1	1	1	1	1	1
611025c	1	1	1	1	1	1	1	1	1	1	1	1
66b86bf	1	1	1	1	1	1	1	1	1	1	1	1

In [7]:
```
#explore any missing years for test set
pd.set_option('display.max_rows',100)
table2 = pd.crosstab(submission['country_code'],submission['year'])
display(table2)
```

year	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011
country_code												
0c2cb01	1	1	1	1	1	1	1	1	1	1	1	1
1043682	1	1	1	1	1	1	1	1	1	1	1	1
14ad63a	1	1	1	1	1	1	1	1	1	1	1	1
1672f60	1	1	1	1	1	1	1	1	1	1	1	1
1af00b8	1	1	1	1	1	1	1	1	1	1	1	1
2cbebc5	1	1	1	1	1	1	1	1	1	1	1	1
2d273d8	1	1	1	1	1	1	1	1	1	1	1	1
364e5f6	1	1	1	1	1	1	1	1	1	1	1	1
4.26E+54	1	1	1	1	1	1	1	1	1	1	1	1
4678a57	1	1	1	1	1	1	1	1	1	1	1	1
4691492	1	1	1	1	1	1	1	1	1	1	1	1
4bc2f1c	1	1	1	1	1	1	1	1	1	1	1	1
640aed4	1	1	1	1	1	1	1	1	1	1	1	1
69d0a7b	1	1	1	1	1	1	1	1	1	1	1	1
74e1891	1	1	1	1	1	1	1	1	1	1	1	1
7d94977	1	1	1	1	1	1	1	1	1	1	1	1
7eb5655	1	1	1	1	1	1	1	1	1	1	1	1
8191231	1	1	1	1	1	1	1	1	1	1	1	1
8220417	1	1	1	1	1	1	1	1	1	1	1	1
881f7fc	1	1	1	1	1	1	1	1	1	1	1	1
9555664	1	1	1	1	1	1	1	1	1	1	1	1
a564371	1	1	1	1	1	1	1	1	1	1	1	1
aa8057e	1	1	1	1	1	1	1	1	1	1	1	1
b0772b9	1	1	1	1	1	1	1	1	1	1	0	1
b1cb8ea	1	1	1	1	1	1	1	1	1	1	1	1
b5f9221	1	1	1	1	1	1	1	1	1	1	1	1
b675f5d	1	1	1	1	1	1	1	1	1	1	1	1
bfb0c8d	1	1	1	1	1	1	1	1	1	1	1	1
c3ce4bf	1	1	1	1	1	1	1	1	1	1	1	1
c5b2a6d	1	1	1	1	1	1	1	1	1	1	1	1
c72199a	1	1	1	1	1	1	1	1	1	1	1	1

In [8]:

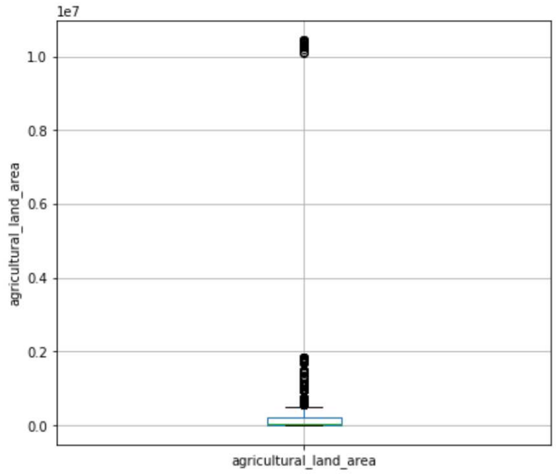
```
# let's visualise the values of the discrete variables
discrete = []
for var in numerical:
    if len(df[var].unique())<20:
        print(var, ' values: ', df[var].unique())
        discrete.append(var)

print('There are {} discrete variables'.format(len(discrete)))
```

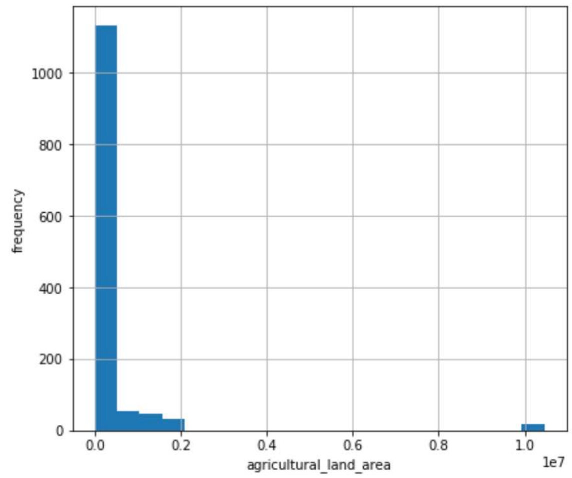
```
year  values:  [2000 2001 2002 2003 2004 2005 2006 2007 2008 2009
2010 2011 2012 2013
 2014 2015]
There are 1 discrete variables
```

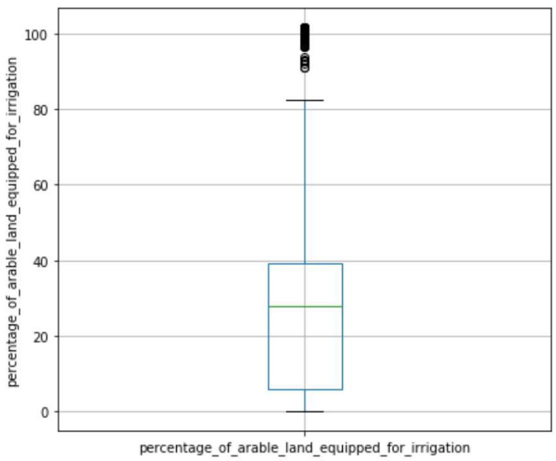
In [9]:
```
continuous = [var for var in numerical if var not in discrete and va
r not in ['row_id', 'prevalence_of_undernourishment']]
continuous
```

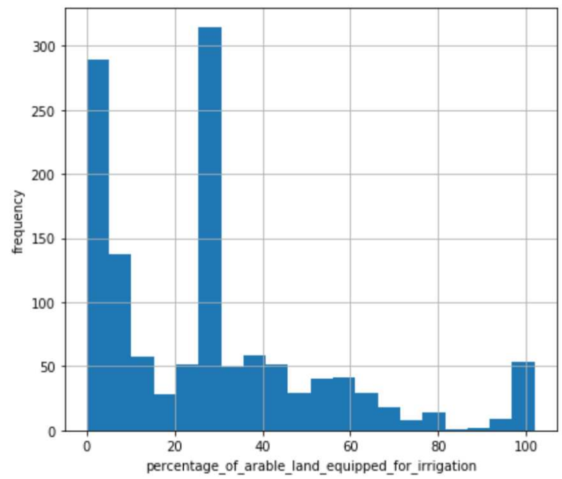
Out[9]:
```
['agricultural_land_area',
 'percentage_of_arable_land_equipped_for_irrigation',
 'cereal_yield',
 'droughts_floods_extreme_temps',
 'forest_area',
 'total_land_area',
 'fertility_rate',
 'life_expectancy',
 'rural_population',
 'total_population',
 'urban_population',
 'population_growth',
 'avg_value_of_food_production',
 'cereal_import_dependency_ratio',
 'food_imports_as_share_of_merch_exports',
 'gross_domestic_product_per_capita_ppp',
 'imports_of_goods_and_services',
 'inequality_index',
 'net_oda_received_percent_gni',
 'net_oda_received_per_capita',
 'tax_revenue_share_gdp',
 'trade_in_services',
 'per_capita_food_production_variability',
 'per_capita_food_supply_variability',
 'adult_literacy_rate',
 'school_enrollment_rate_female',
 'school_enrollment_rate_total',
 'avg_supply_of_protein_of_animal_origin',
 'caloric_energy_from_cereals_roots_tubers',
 'access_to_improved_sanitation',
 'access_to_improved_water_sources',
 'anemia_prevalence',
 'obesity_prevalence',
 'open_defecation',
 'hiv_incidence',
 'rail_lines_density',
 'access_to_electricity',
 'co2_emissions',
 'unemployment_rate',
 'total_labor_force',
 'military_expenditure_share_gdp',
 'proportion_of_seats_held_by_women_in_gov',
 'political_stability']
```

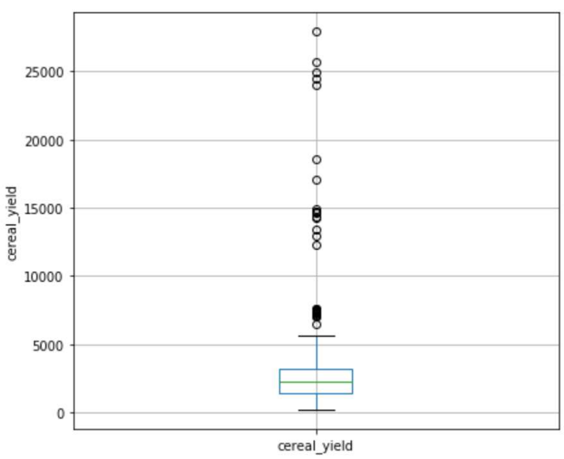
In [10]:
```
# let's make boxplots to visualise outliers in the continuous variab
les
# and histograms to get an idea of the distribution

for var in continuous:
    plt.figure(figsize=(15,6))
    plt.subplot(1, 2, 1)
    fig = df.boxplot(column=var)
    fig.set_title('')
    fig.set_ylabel(var)

    plt.subplot(1, 2, 2)
    fig = df[var].hist(bins=20)
    fig.set_ylabel('frequency')
    fig.set_xlabel(var)

    plt.show()
```

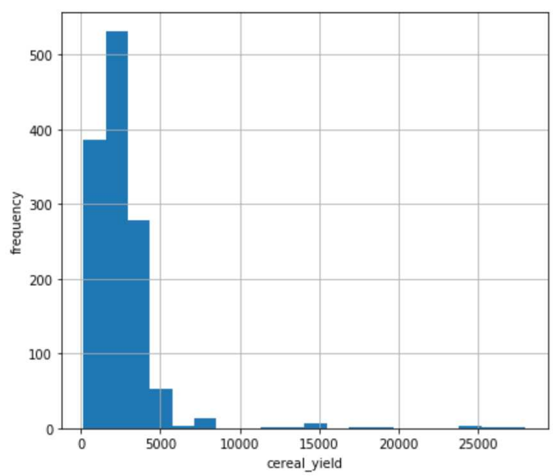

1e7
1.0
0.8
0.6
0.4
0.2
0.0
agricultural_land_area
agricultural_land_area

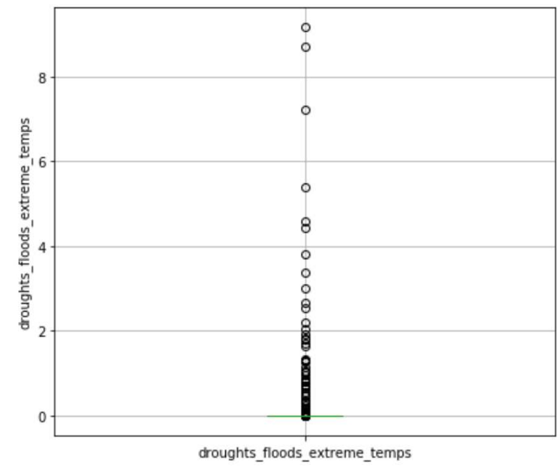

1000
800
600
400
200
0
frequency
0.0
0.2
0.4
0.6
0.8
1.0
1e7
agricultural_land_area

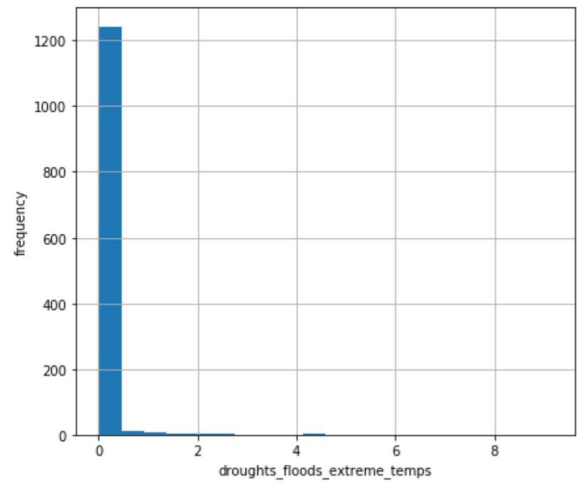

100
80
60
40
20
0
percentage_of_arable_land_equipped_for_irrigation
percentage_of_arable_land_equipped_for_irrigation

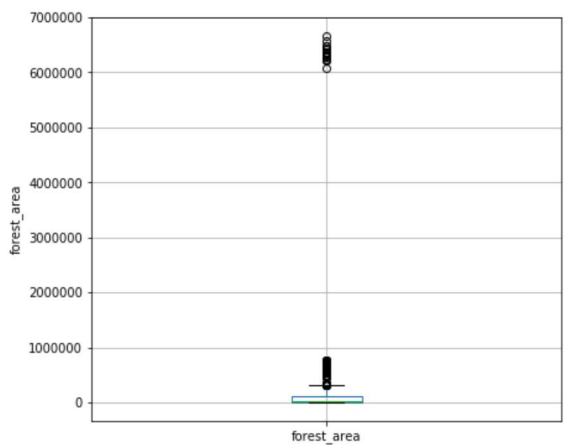

300
250
200
150
100
50
0
frequency
0
20
40
60
80
100
percentage_of_arable_land_equipped_for_irrigation

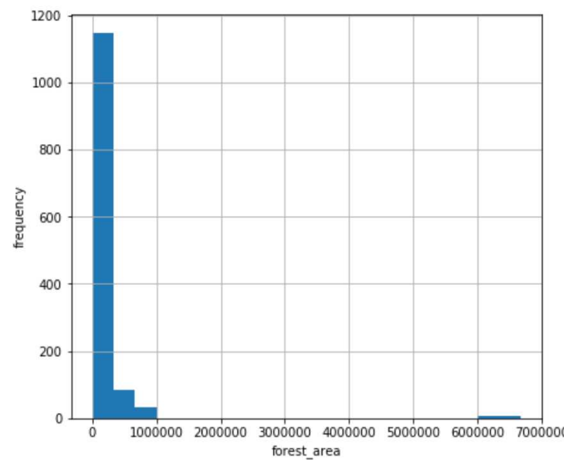

25000
20000
15000
10000
5000
0
cereal_yield
cereal_yield

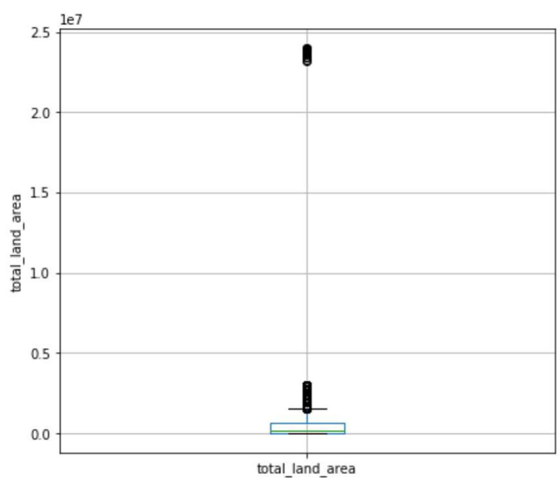

500
400
300
200
100
0
frequency
0
5000
10000
15000
20000
25000
cereal_yield

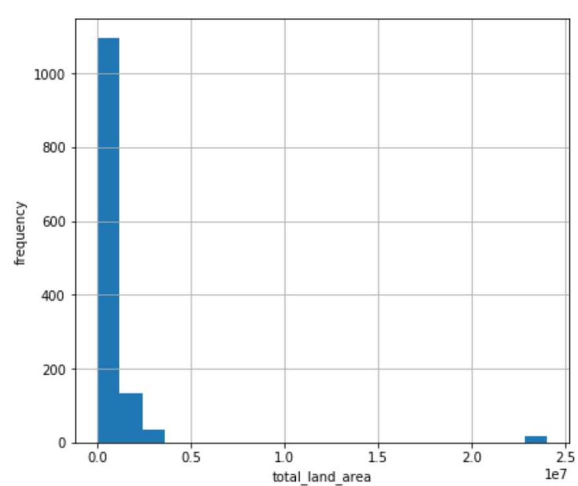

droughts_floods_extreme_temps
droughts_floods_extreme_temps

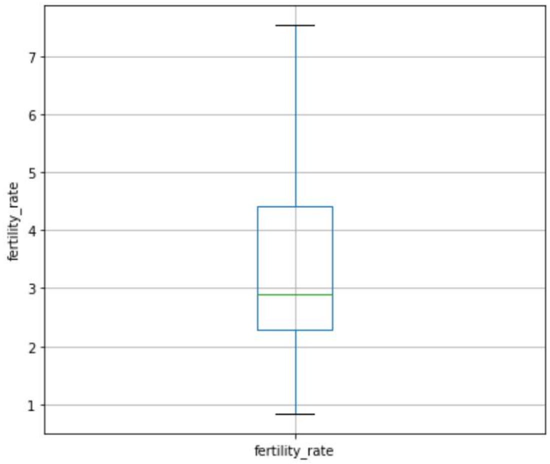

frequency
droughts_floods_extreme_temps

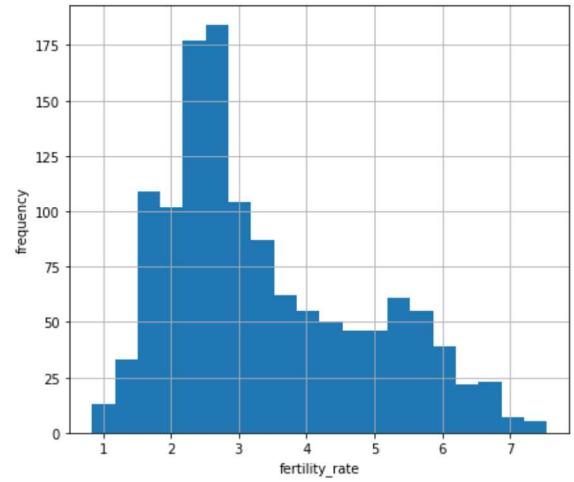

forest_area
forest_area

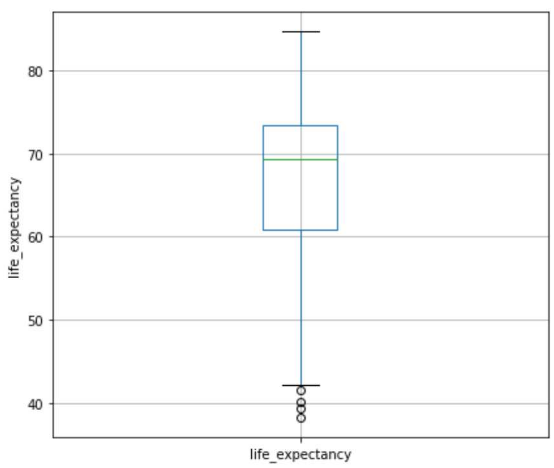

frequency
forest_area

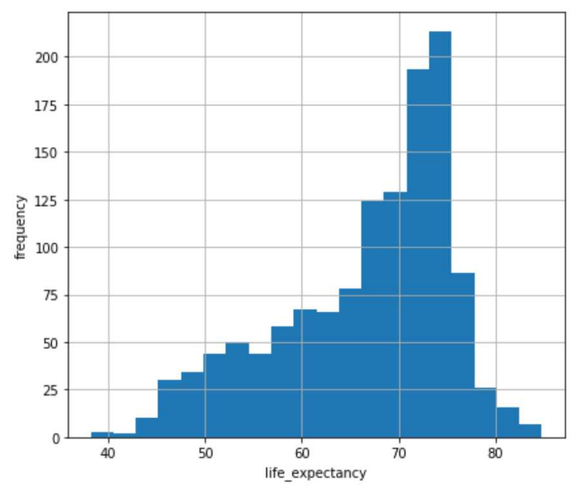

1e7
total_land_area
total_land_area

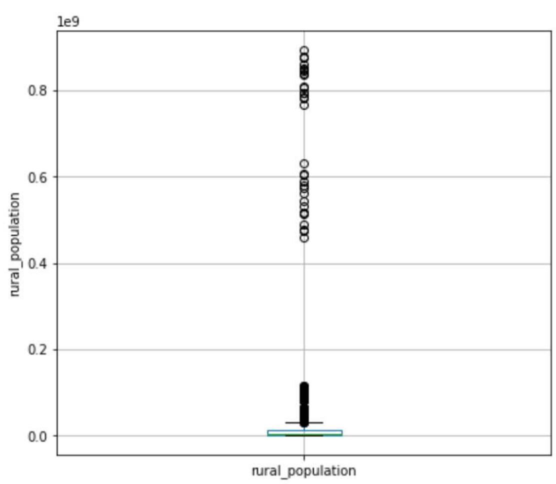

frequency
total_land_area
1e7

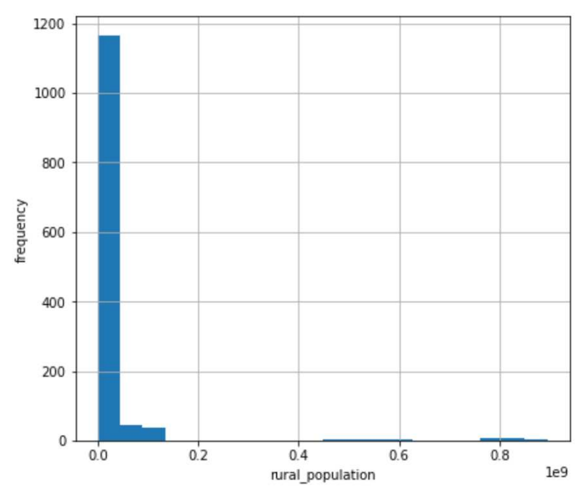

fertility_rate
fertility_rate

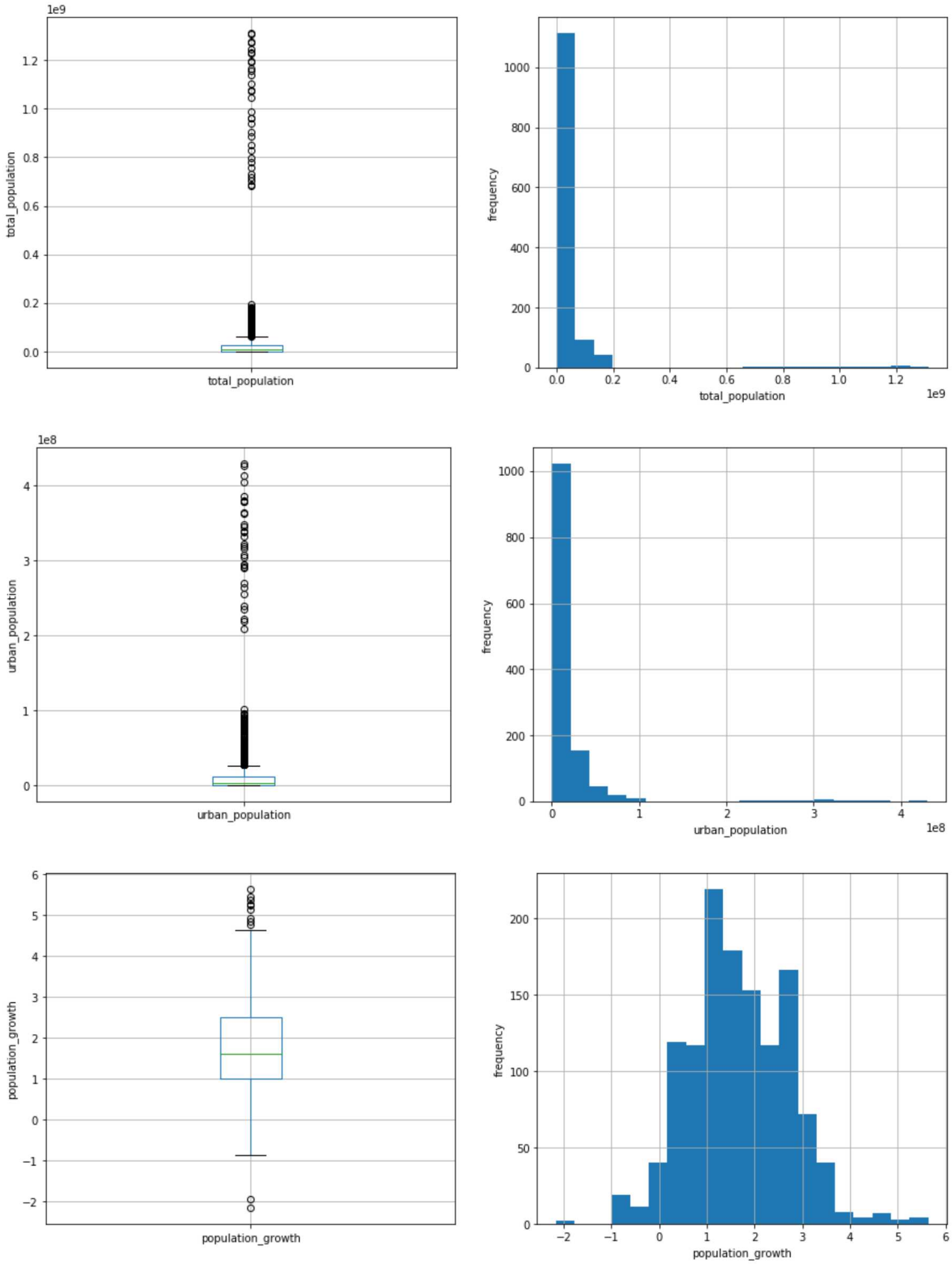

frequency
fertility_rate

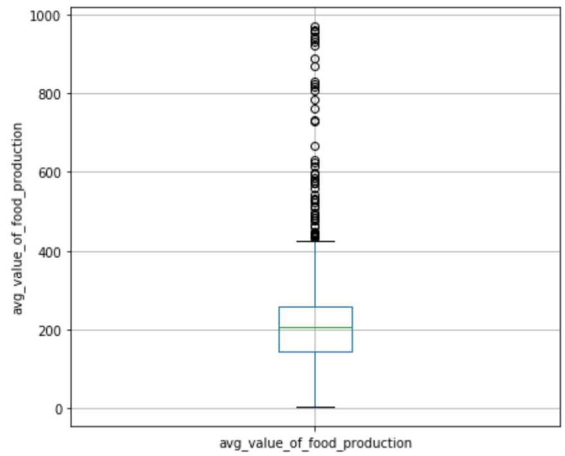

life_expectancy
life_expectancy

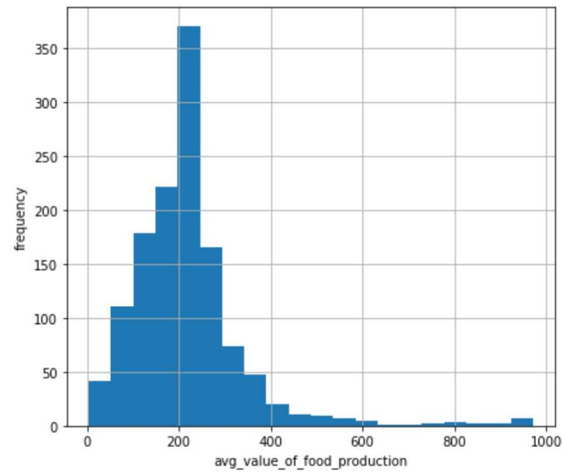

frequency
life_expectancy

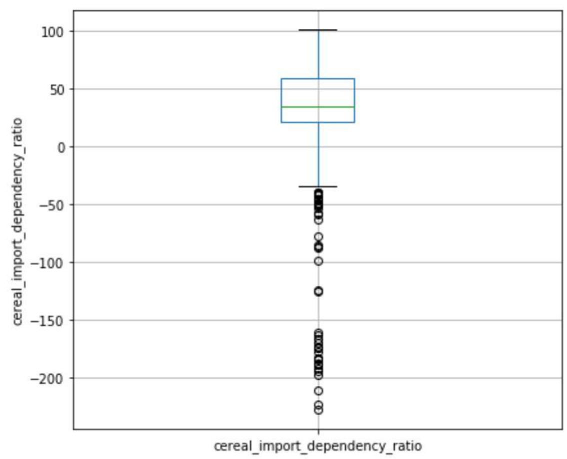

1e9
rural_population
rural_population

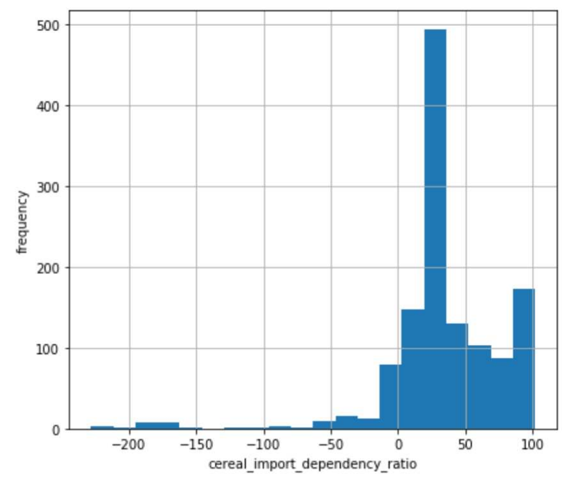

frequency
rural_population
1e9

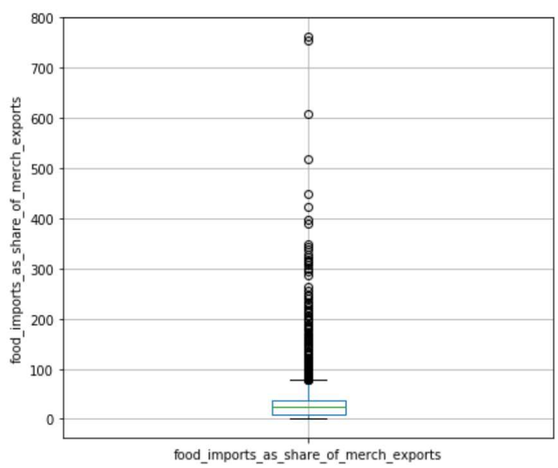

1e9
total_population
frequency
total_population
1e9
1e8
urban_population
frequency
urban_population
1e8
population_growth
frequency
population_growth

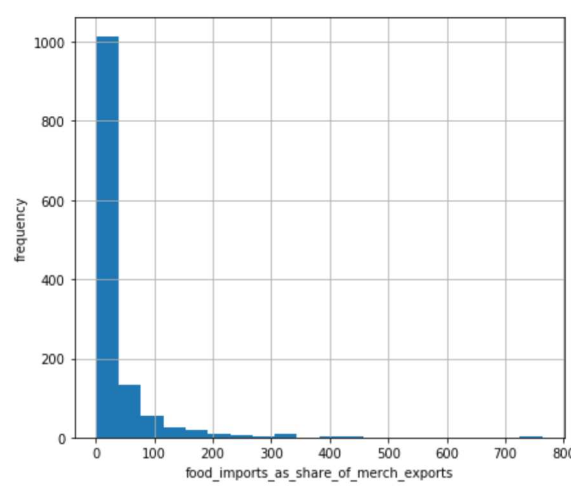

avg_value_of_food_production
avg_value_of_food_production

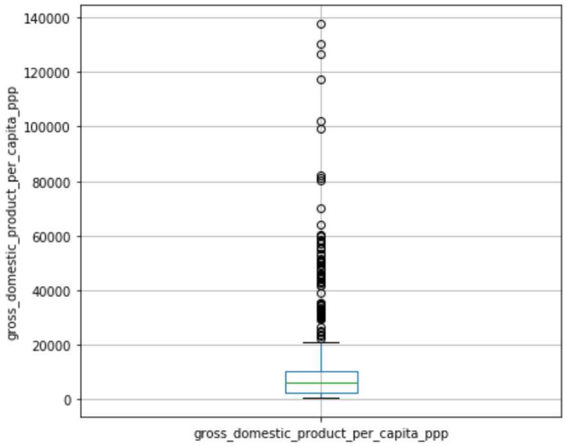

frequency
avg_value_of_food_production

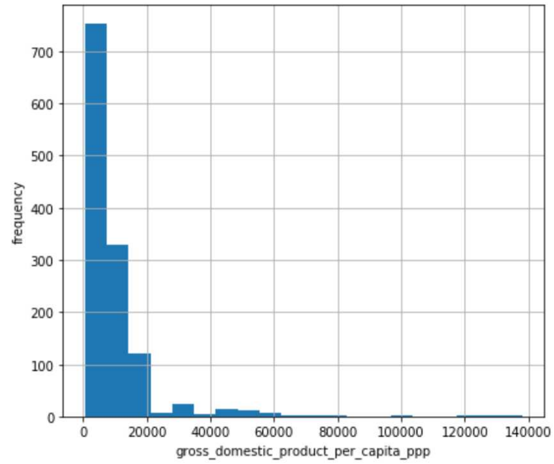

cereal_import_dependency_ratio
cereal_import_dependency_ratio

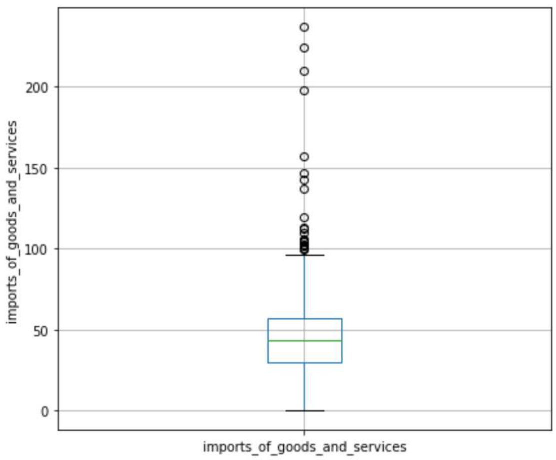

frequency
cereal_import_dependency_ratio

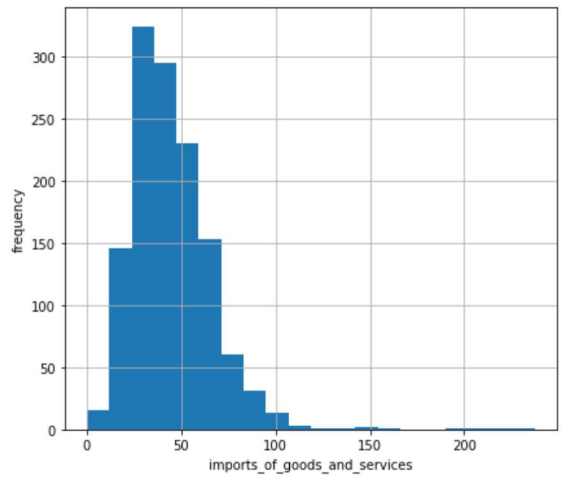

food_imports_as_share_of_merch_exports
food_imports_as_share_of_merch_exports

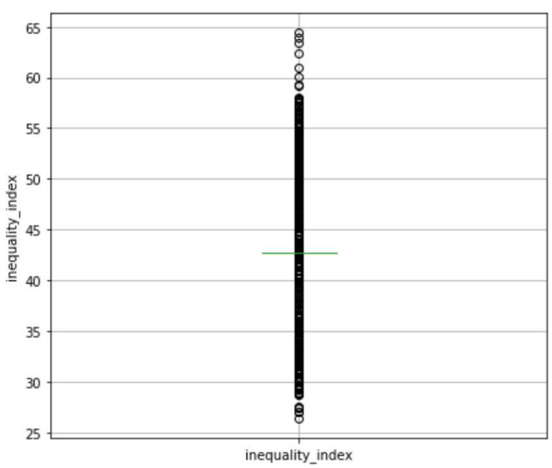

frequency
food_imports_as_share_of_merch_exports

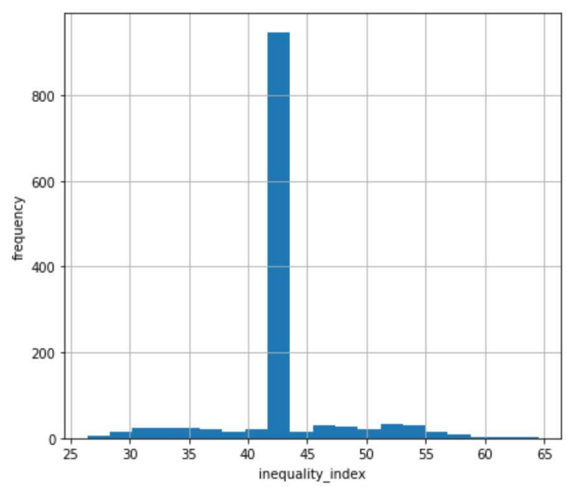

140000
120000
100000
80000
60000
40000
20000
0
gross_domestic_product_per_capita_ppp
gross_domestic_product_per_capita_ppp

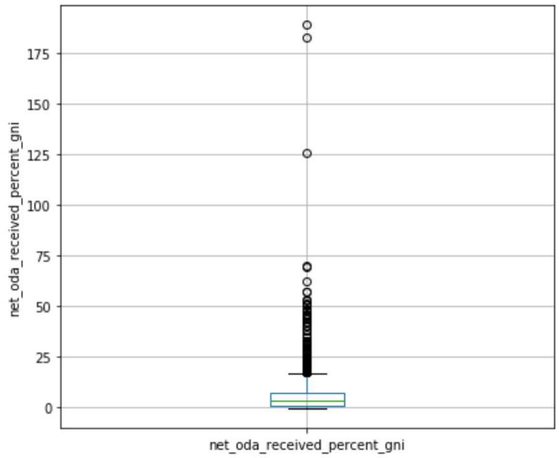

700
600
500
400
300
200
100
0
frequency
0
20000
40000
60000
80000
100000
120000
140000
gross_domestic_product_per_capita_ppp


200
150
100
50
0
imports_of_goods_and_services
imports_of_goods_and_services

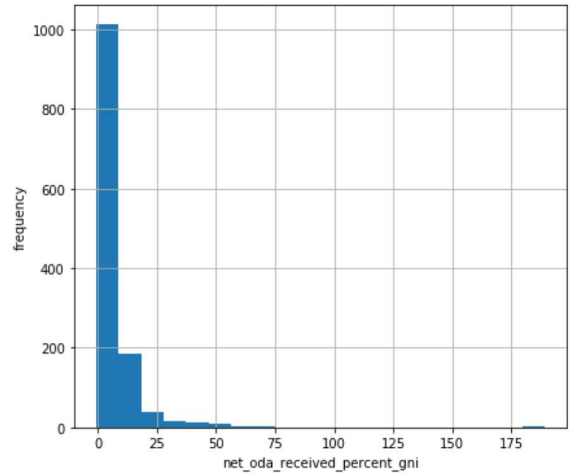

300
250
200
150
100
50
0
frequency
0
50
100
150
200
imports_of_goods_and_services


65
60
55
50
45
40
35
30
25
inequality_index
inequality_index

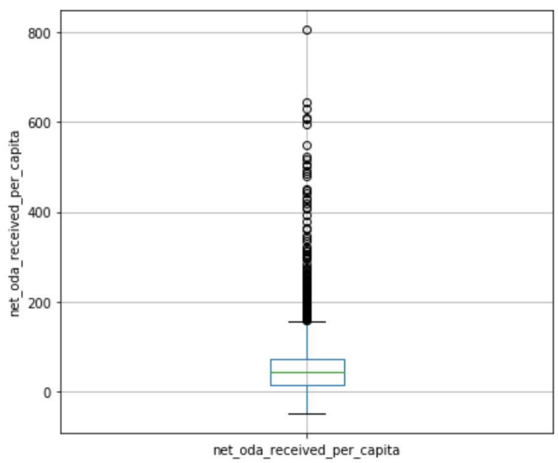

800
600
400
200
0
frequency
25
30
35
40
45
50
55
60
65
inequality_index



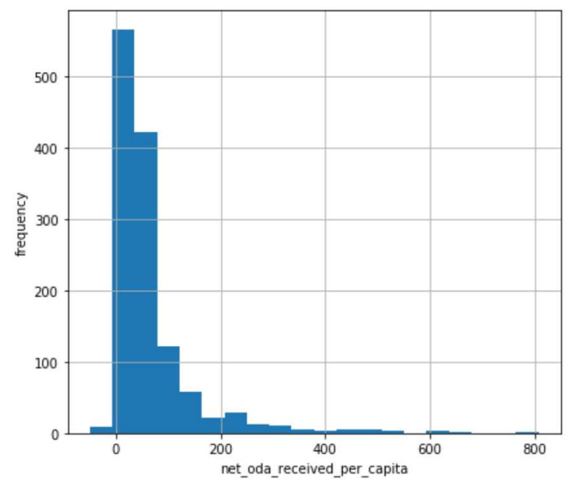
net_oda_received_percent_gni
0
25
50
75
100
125
150
175
net_oda_received_percent_gni



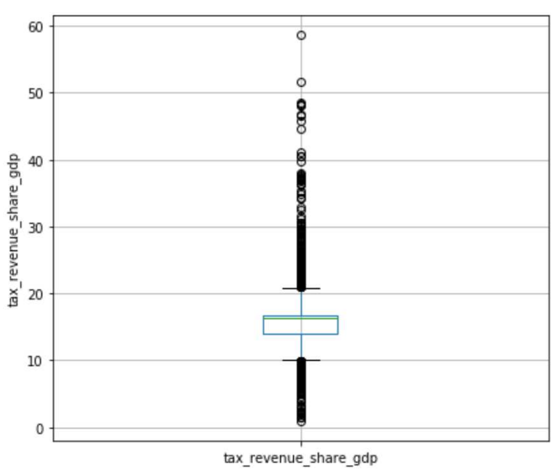
frequency
0
200
400
600
800
1000
0
25
50
75
100
125
150
175
net_oda_received_percent_gni



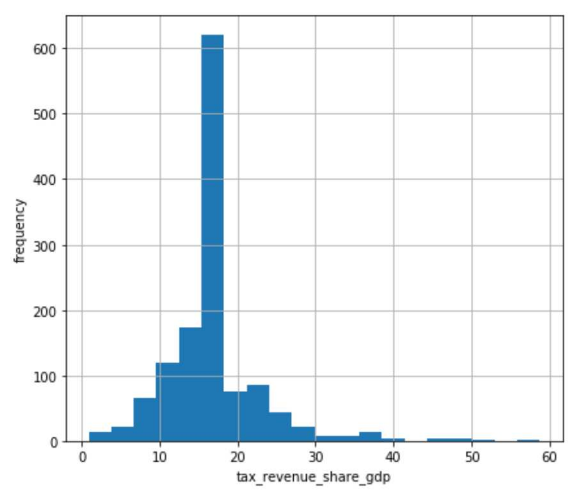
net_oda_received_per_capita
0
200
400
600
800
net_oda_received_per_capita



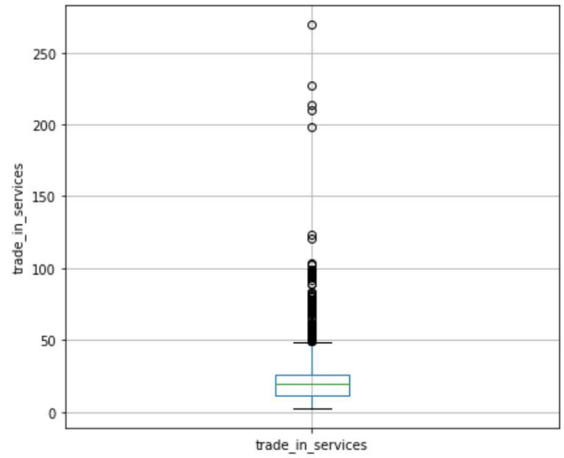
frequency
0
100
200
300
400
500
0
200
400
600
800
net_oda_received_per_capita

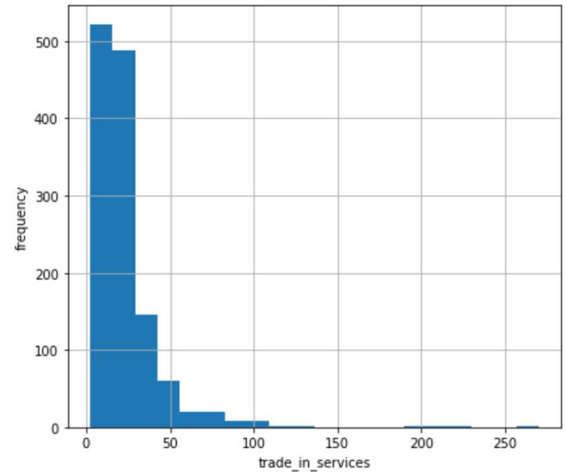


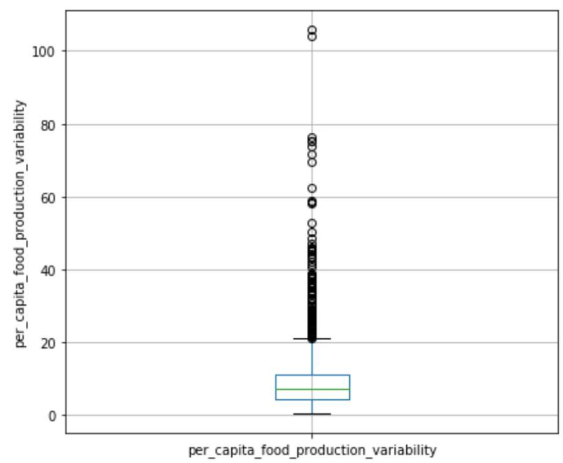
tax_revenue_share_gdp
0
10
20
30
40
50
60
tax_revenue_share_gdp

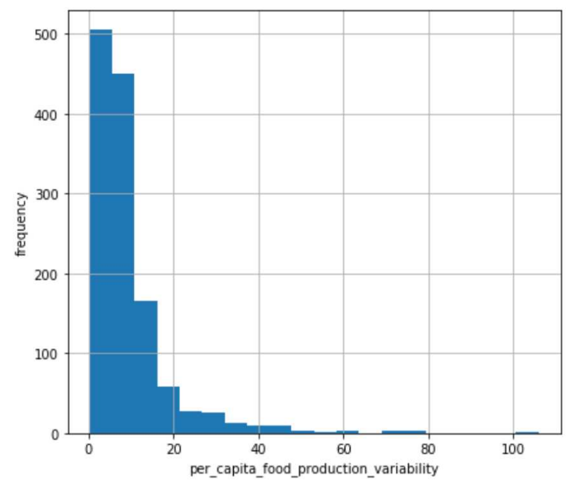


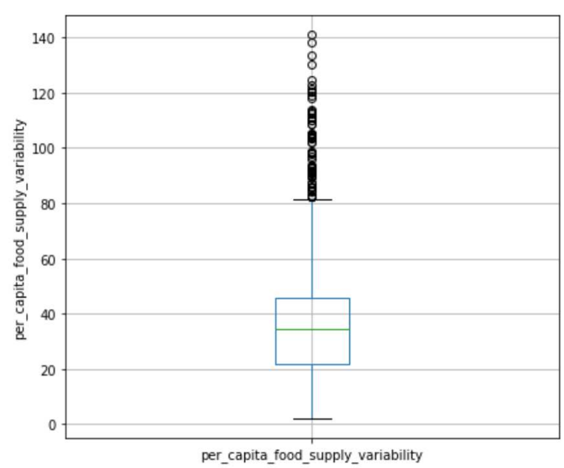
frequency
0
100
200
300
400
500
600
0
10
20
30
40
50
60
tax_revenue_share_gdp

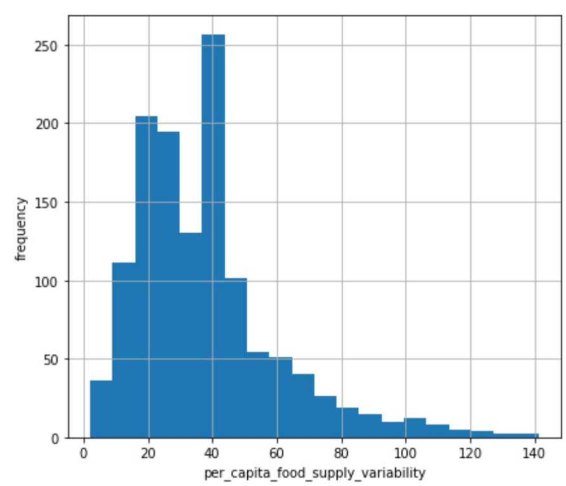

trade_in_services
0
50
100
150
200
250

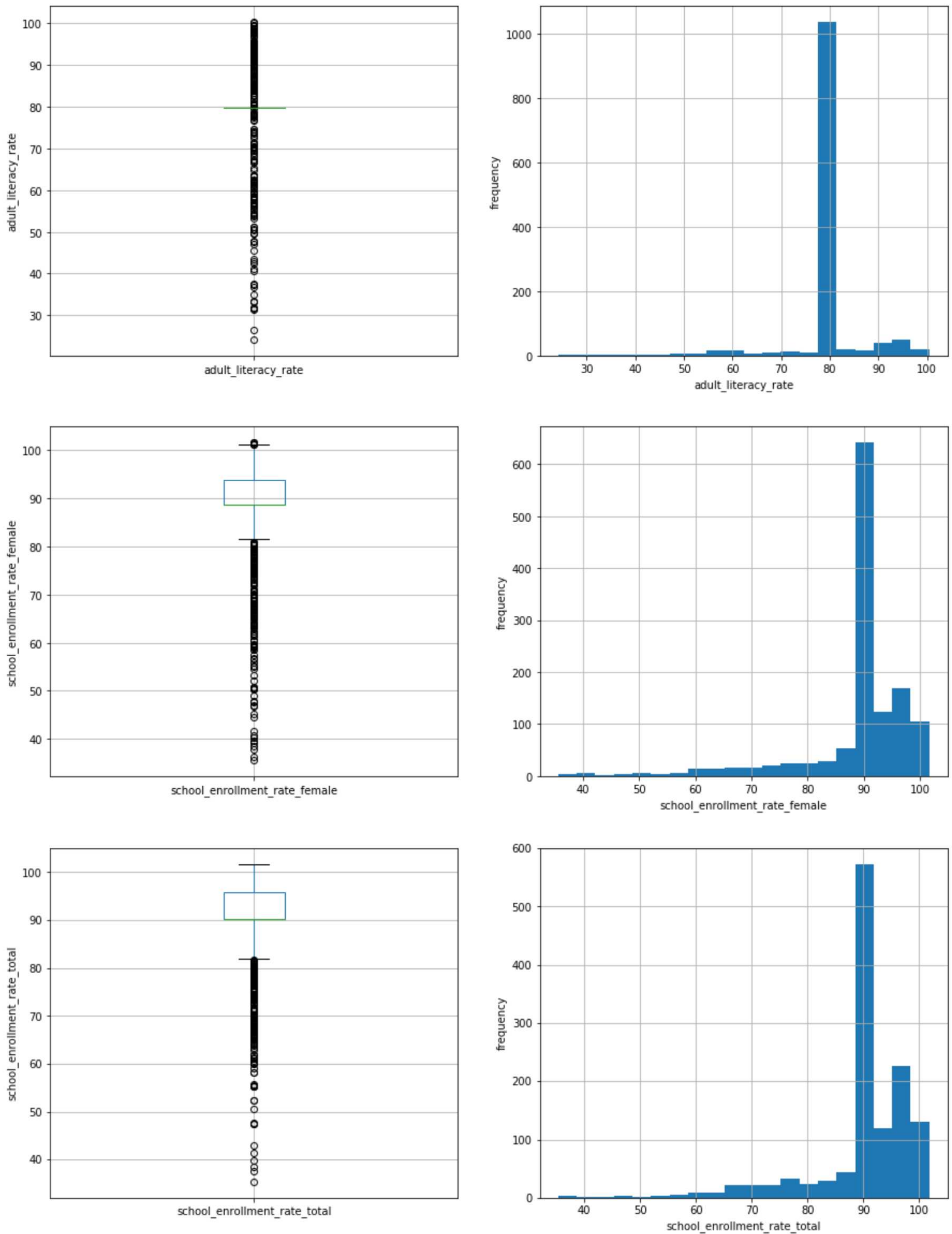

frequency
trade_in_services
0
100
200
300
400
500
0
50
100
150
200
250

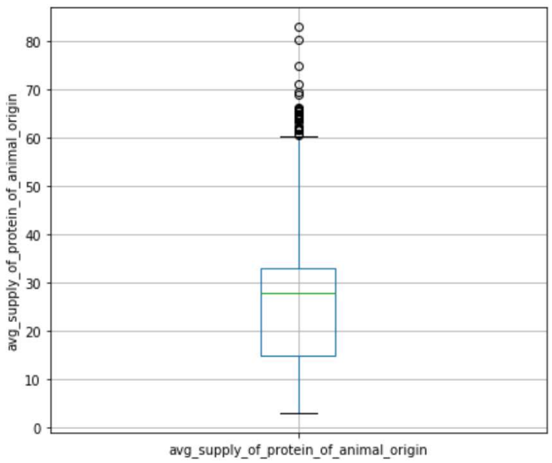
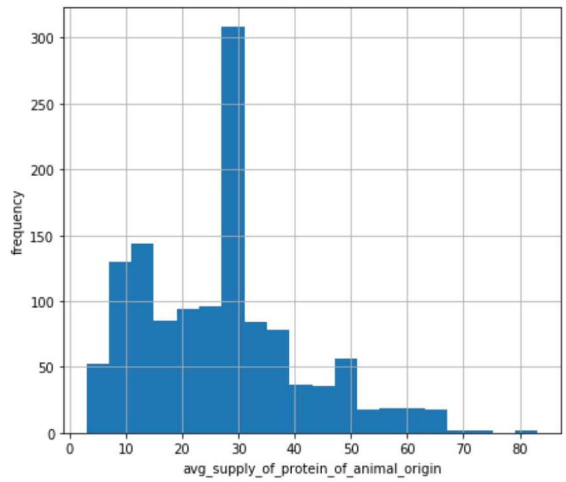
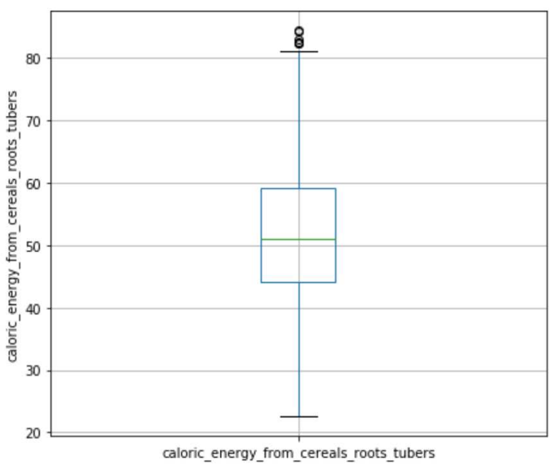
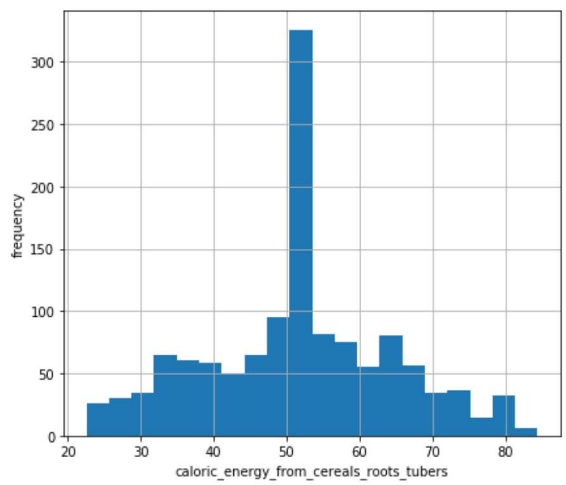
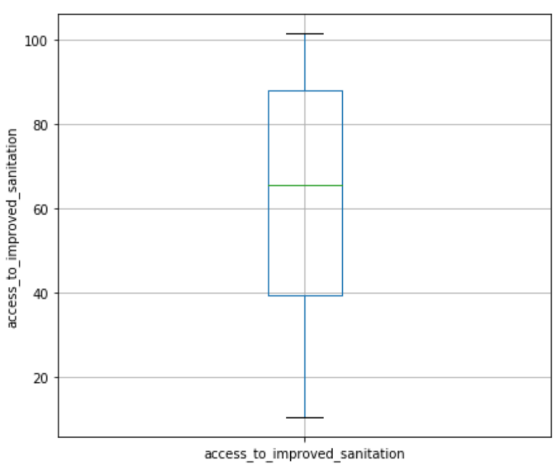
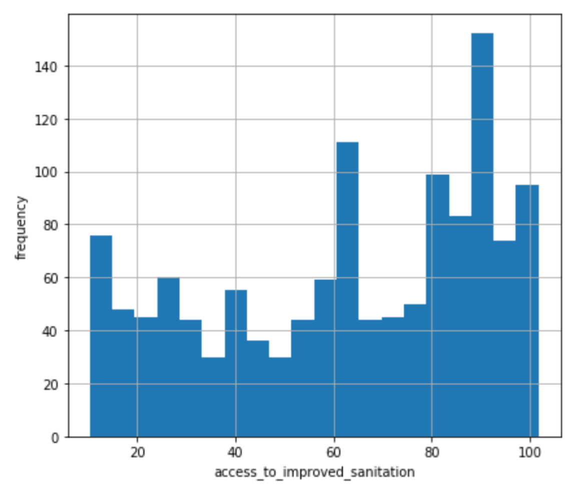

per_capita_food_production_variability
0
20
40
60
80
100

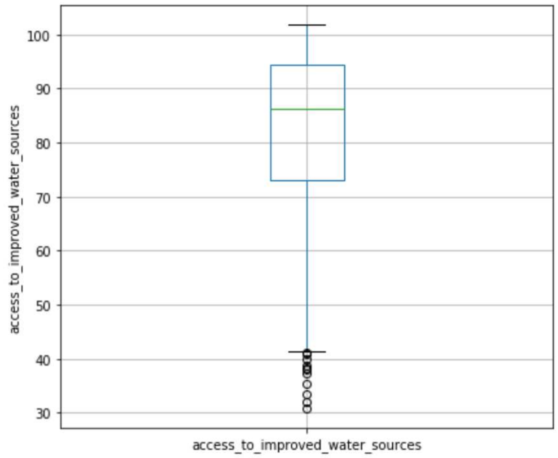

frequency
per_capita_food_production_variability
0
100
200
300
400
500
0
20
40
60
80
100

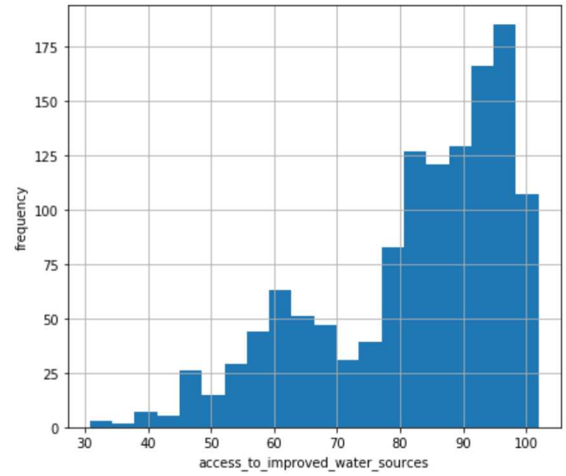

per_capita_food_supply_variability
0
20
40
60
80
100
120
140

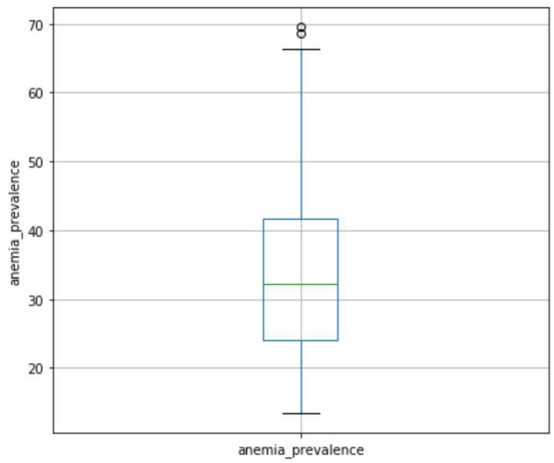

frequency
per_capita_food_supply_variability
0
50
100
150
200
250
0
20
40
60
80
100
120
140

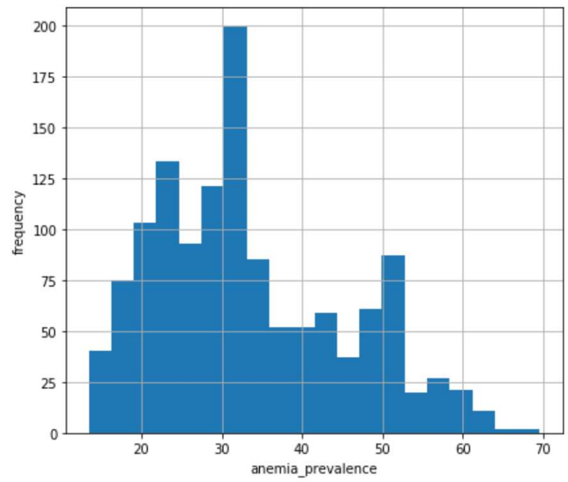

adult_literacy_rate
frequency
adult_literacy_rate
school_enrollment_rate_female
frequency
school_enrollment_rate_female
school_enrollment_rate_total
frequency
school_enrollment_rate_total

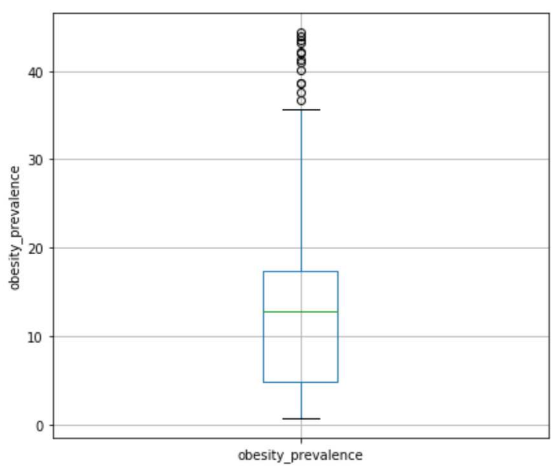

avg_supply_of_protein_of_animal_origin
avg_supply_of_protein_of_animal_origin

frequency
avg_supply_of_protein_of_animal_origin

caloric_energy_from_cereals_roots_tubers
caloric_energy_from_cereals_roots_tubers

frequency
caloric_energy_from_cereals_roots_tubers

access_to_improved_sanitation
access_to_improved_sanitation

frequency
access_to_improved_sanitation

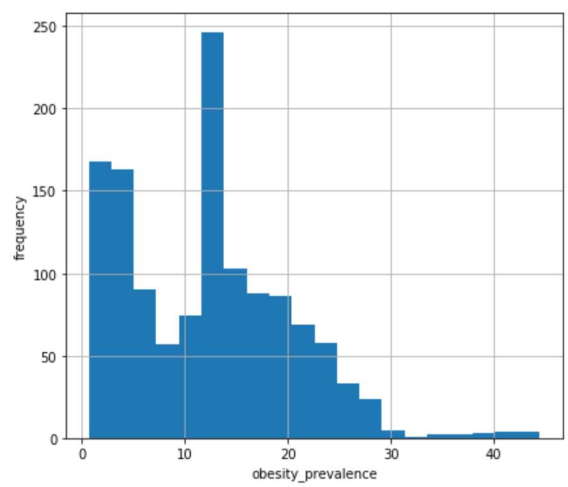

access_to_improved_water_sources
30
40
50
60
70
80
90
100
access_to_improved_water_sources

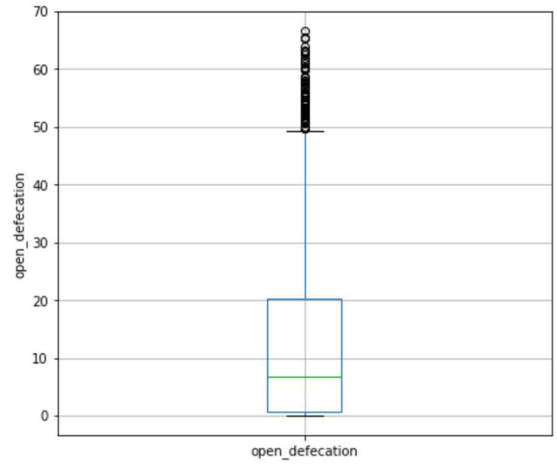

frequency
0
25
50
75
100
125
150
175
30
40
50
60
70
80
90
100
access_to_improved_water_sources

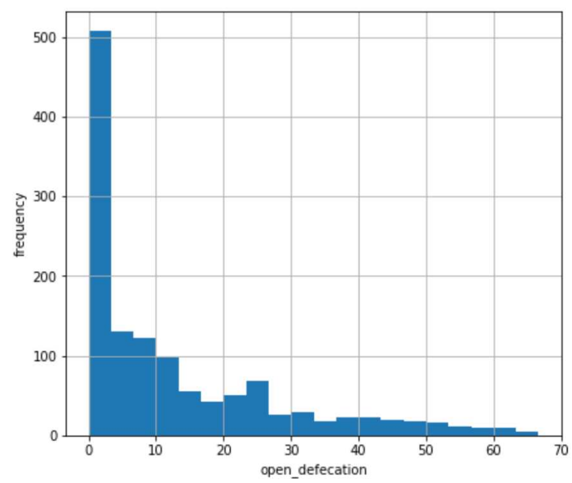

anemia_prevalence
20
30
40
50
60
70
anemia_prevalence

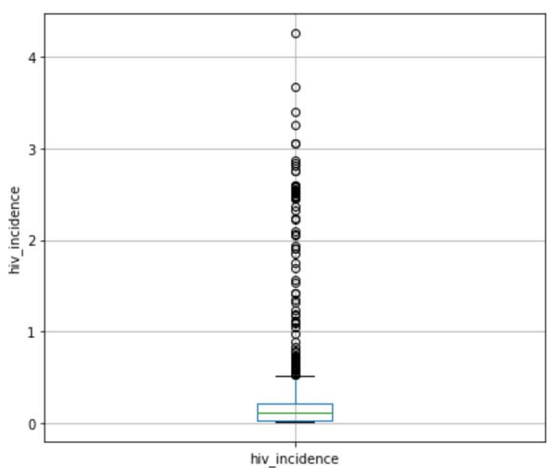

frequency
0
25
50
75
100
125
150
175
200
20
30
40
50
60
70
anemia_prevalence

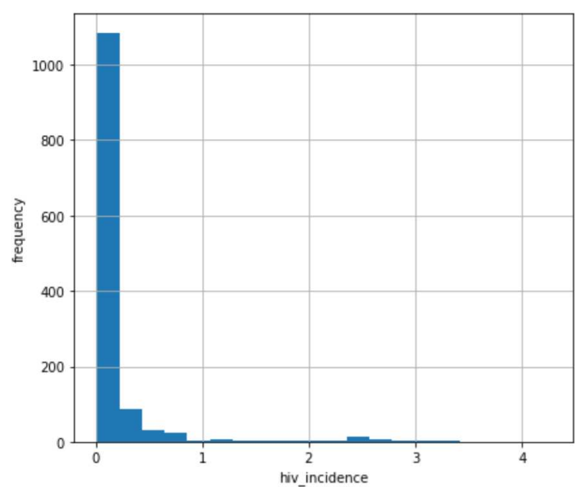

obesity_prevalence
0
10
20
30
40
obesity_prevalence

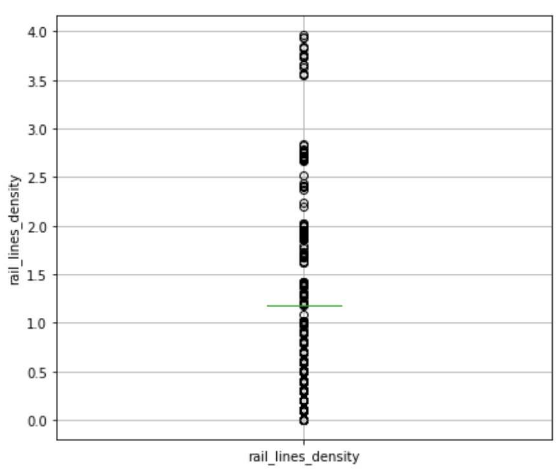

frequency
0
50
100
150
200
250
0
10
20
30
40
obesity_prevalence

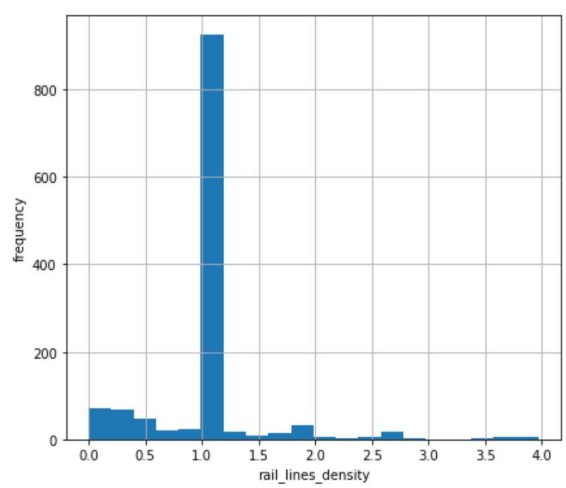

open_defecation
0
10
20
30
40
50
60
70
open_defecation

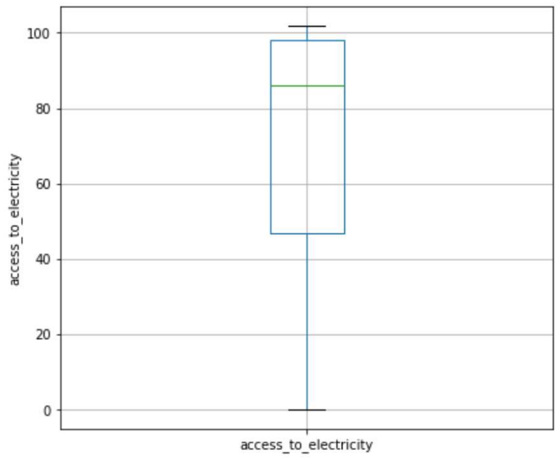

frequency
0
100
200
300
400
500
0
10
20
30
40
50
60
70
open_defecation

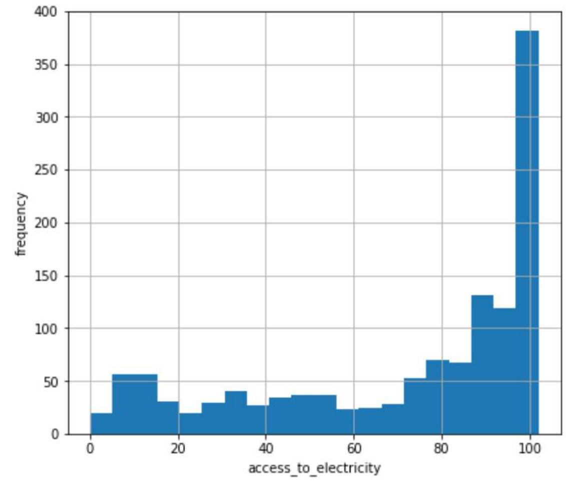

hiv_incidence
0
1
2
3
4
hiv_incidence

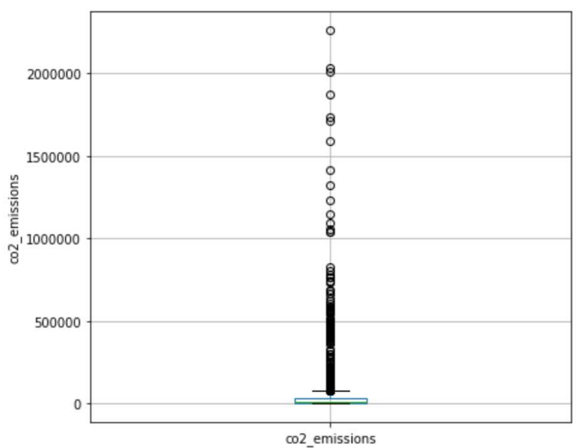

frequency
0
200
400
600
800
1000
0
1
2
3
4
hiv_incidence

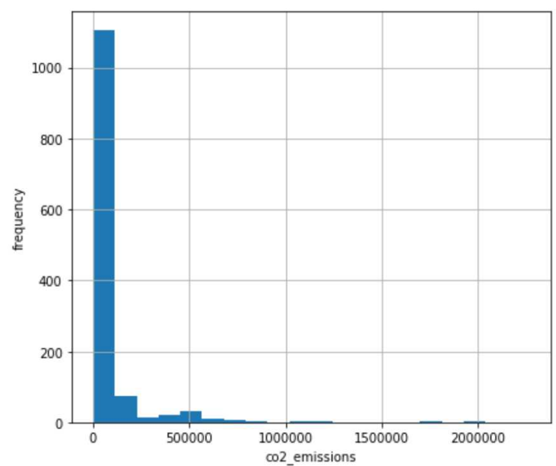

rail_lines_density
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
rail_lines_density

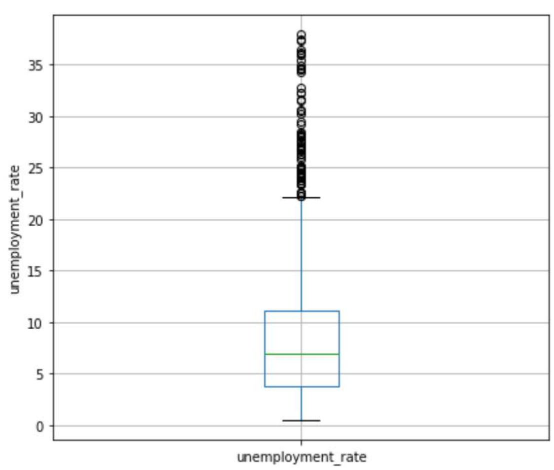

frequency
0
200
400
600
800
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
rail_lines_density

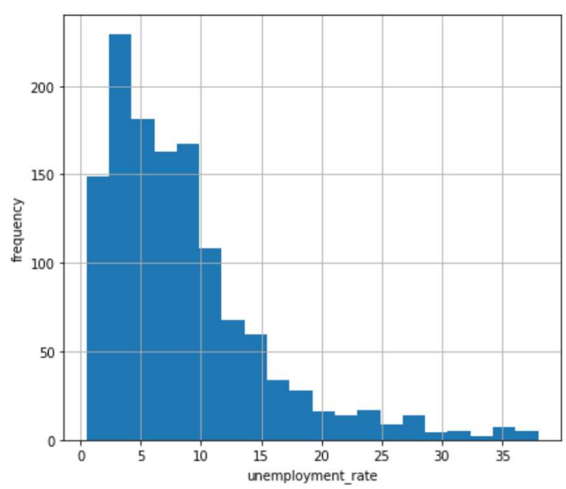

access_to_electricity
access_to_electricity

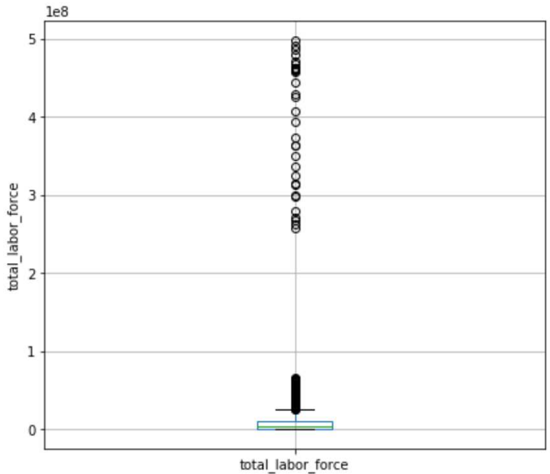

frequency
access_to_electricity

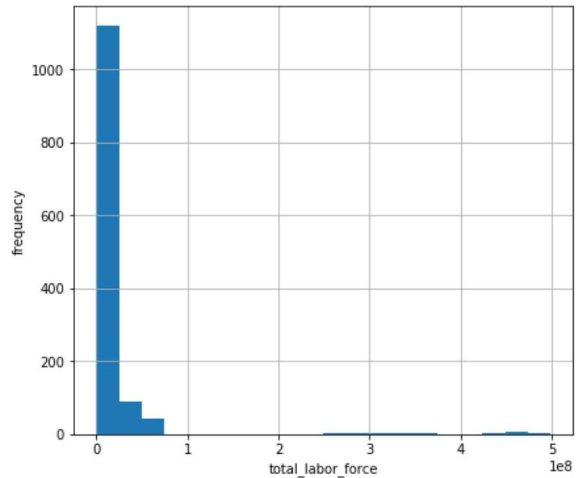

co2_emissions
co2_emissions

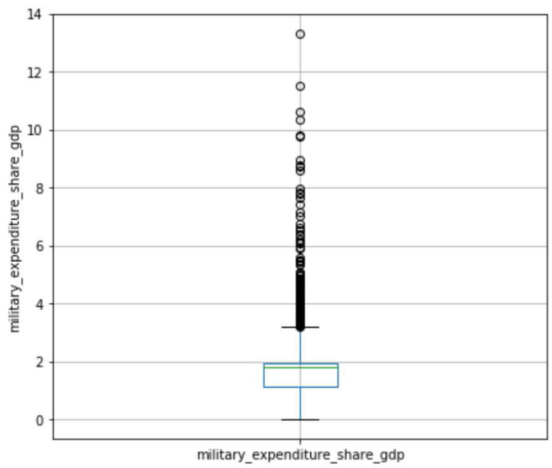

frequency
co2_emissions

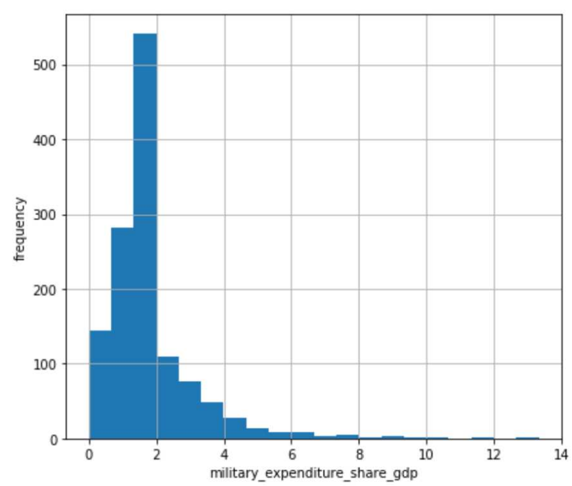

unemployment_rate
unemployment_rate

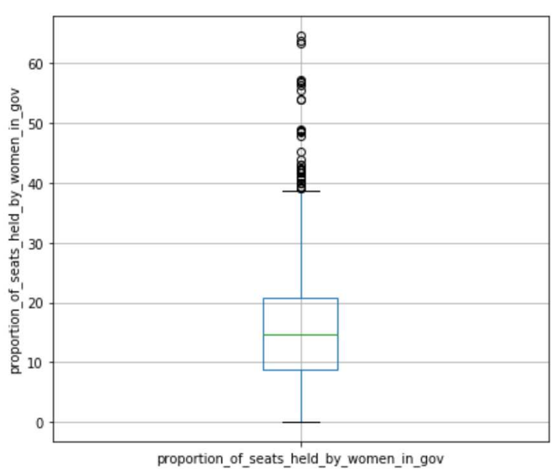

frequency
unemployment_rate

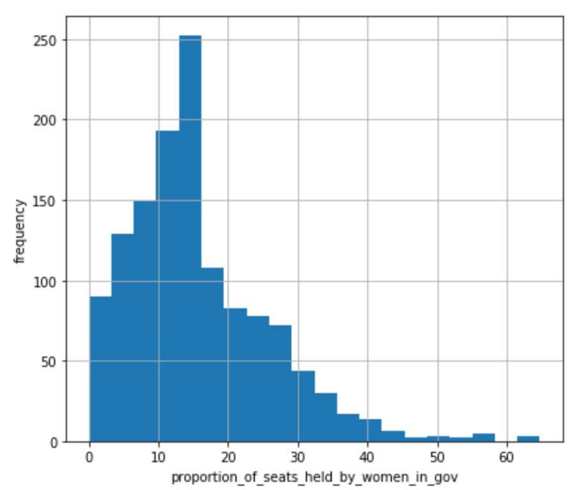

1e8
total_labor_force
total_labor_force

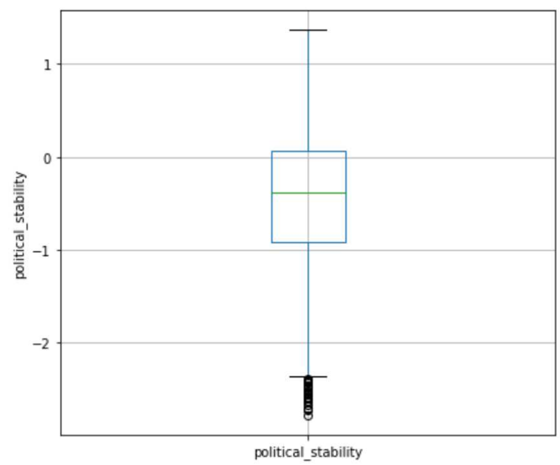

frequency
total_labor_force
1e8

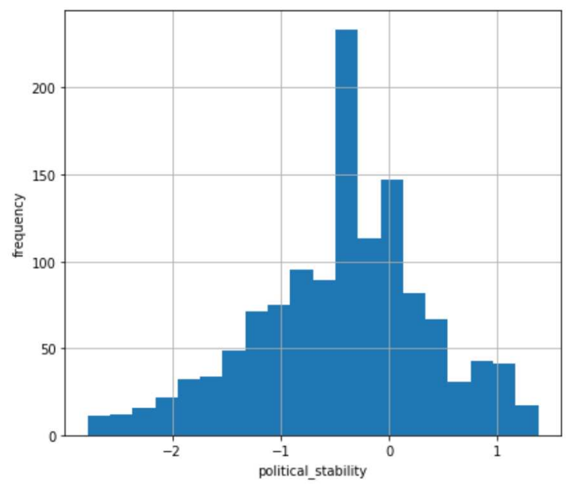

military_expenditure_share_gdp
military_expenditure_share_gdp


frequency
military_expenditure_share_gdp


proportion_of_seats_held_by_women_in_gov
proportion_of_seats_held_by_women_in_gov


frequency
proportion_of_seats_held_by_women_in_gov





In [11]:
```
# outliers in discrete variables
for var in discrete:
    print(df[var].value_counts() / np.float(len(df)))
    print()
```

```
2015    0.0625
2014    0.0625
2013    0.0625
2012    0.0625
2011    0.0625
2010    0.0625
2009    0.0625
2008    0.0625
2007    0.0625
2006    0.0625
2005    0.0625
2004    0.0625
2003    0.0625
2002    0.0625
2001    0.0625
2000    0.0625
Name: year, dtype: float64
```

In [12]:
```
for var in categorical:
    print(var, ' contains ', len(df[var].unique()), ' labels')
```

```
country_code  contains  80  labels
```

Separate train and test set

In [13]:
```
# Let's separate into train and test set

X_train, X_test, y_train, y_test = train_test_split(df, df.prevalenc
e_of_undernourishment, test_size=0.2, random_state=0)
X_train.shape, X_test.shape
```

Out[13]: ((1024, 47), (256, 47))

Outliers in Numerical variables

In order to tackle outliers and skewed distributions at the same time, I suggested I would do discretisation. And in order to find the optimal buckets automatically, I would use decision trees to find the buckets for me.

In [14]:
```
def tree_binariser(var):
    score_ls = [] # here I will store the mse

    for tree_depth in [1,2,3,4]:
        # call the model
        tree_model = DecisionTreeRegressor(max_depth=tree_depth)

        # train the model using 3 fold cross validation
        scores = cross_val_score(tree_model, X_train[var].to_frame()
, y_train, cv=3, scoring='neg_mean_squared_error')
        score_ls.append(np.mean(scores))

    # find depth with smallest mse
    depth = [1,2,3,4][np.argmax(score_ls)]
    #print(score_ls, np.argmax(score_ls), depth)

    # transform the variable using the tree
    tree_model = DecisionTreeRegressor(max_depth=depth)
    tree_model.fit(X_train[var].to_frame(), X_train.prevalence_of_un
dernourishment)
    X_train[var] = tree_model.predict(X_train[var].to_frame())
    X_test[var] = tree_model.predict(X_test[var].to_frame())
    submission[var] =  tree_model.predict(submission[var].to_frame()
)
```

In [15]:
```
for var in continuous:
    tree_binariser(var)
```

In [16]:
```
X_train[continuous].head()
```

Out[16]:

	agricultural_land_area	percentage_of_arable_land_equipped_for_irrigation	cerea
308	15.467345	14.406020	12.80
295	15.467345	14.406020	7.866
915	15.467345	14.406020	12.80
465	26.980645	18.339051	17.40
298	15.467345	14.406020	7.866

In [17]:
```python
for var in continuous:
    print(var, len(X_train[var].unique()))
```

```
agricultural_land_area 16
percentage_of_arable_land_equipped_for_irrigation 4
cereal_yield 8
droughts_floods_extreme_temps 2
forest_area 16
total_land_area 16
fertility_rate 4
life_expectancy 8
rural_population 16
total_population 16
urban_population 8
population_growth 4
avg_value_of_food_production 8
cereal_import_dependency_ratio 16
food_imports_as_share_of_merch_exports 4
gross_domestic_product_per_capita_ppp 8
imports_of_goods_and_services 8
inequality_index 2
net_oda_received_percent_gni 4
net_oda_received_per_capita 8
tax_revenue_share_gdp 2
trade_in_services 8
per_capita_food_production_variability 4
per_capita_food_supply_variability 2
adult_literacy_rate 2
school_enrollment_rate_female 4
school_enrollment_rate_total 4
avg_supply_of_protein_of_animal_origin 8
caloric_energy_from_cereals_roots_tubers 8
access_to_improved_sanitation 4
access_to_improved_water_sources 8
anemia_prevalence 4
obesity_prevalence 8
open_defecation 4
hiv_incidence 8
rail_lines_density 8
access_to_electricity 8
co2_emissions 16
unemployment_rate 2
total_labor_force 8
military_expenditure_share_gdp 2
proportion_of_seats_held_by_women_in_gov 15
political_stability 4
```

Feature scaling

In [18]:
```
X_train.describe()
```

Out[18]:

	row_id	year	agricultural_land_area	percentage_of_arable_land_
count	1024.000000	1024.000000	1024.000000	1024.000000
mean	699.700195	2007.336914	16.422620	16.422620
std	404.644165	4.598309	7.244404	3.614167
min	0.000000	2000.000000	4.617564	14.406020
25%	349.750000	2003.000000	10.044163	14.406020
50%	690.500000	2007.000000	15.467345	14.406020
75%	1039.250000	2011.000000	20.664900	18.339051
max	1400.000000	2015.000000	40.541780	31.866695

In [19]:
```
training_vars = [var for var in X_train.columns if var not in ['row_
id','country_code','year', 'prevalence_of_undernourishment']]
```

In [20]:
```
# fit scaler
scaler = StandardScaler() # create an instance
scaler.fit(X_train[training_vars]) #  fit  the scaler to the train s
et for later use
```

Out[20]:
```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

Machine Learning algorithm building

xgboost

In [21]:
```
xgb_model = xgb.XGBRegressor(max_depth=32,learning_rate=0.01,n_estim
ators=1000,silent=True,)

eval_set = [(X_test[training_vars], y_test)]
xgb_model.fit(X_train[training_vars], y_train, eval_set=eval_set, ve
rbose=False)

pred = xgb_model.predict(X_train[training_vars])
print('xgb train mse: {}'.format(mean_squared_error(y_train, pred)))
pred = xgb_model.predict(X_test[training_vars])
print('xgb test mse: {}'.format(mean_squared_error(y_test, pred)))
```

```
xgb train mse: 0.042008455030963364
xgb test mse: 11.23932270180297
```

Random Forests

```
In [22]: rf_model = RandomForestRegressor(n_estimators=2000,max_depth=32,crit
erion="mse",random_state=1234)
rf_model.fit(X_train[training_vars], y_train)

pred = rf_model.predict(X_train[training_vars])
print('rf train mse: {}'.format(mean_squared_error(y_train, pred)))
pred = rf_model.predict(X_test[training_vars])
print('rf test mse: {}'.format(mean_squared_error(y_test, pred)))
```

```
rf train mse: 0.8891796707427881
rf test mse: 14.013620044019174
```

Support vector machine

```
In [23]: SVR_model = SVR()
SVR_model.fit(scaler.transform(X_train[training_vars]), y_train)

pred = SVR_model.predict(scaler.transform(X_train[training_vars]))
print('SVR train mse: {}'.format(mean_squared_error(y_train, pred)))
pred = SVR_model.predict(scaler.transform(X_test[training_vars]))
print('SVR test mse: {}'.format(mean_squared_error(y_test, pred)))
```

```
SVR train mse: 37.01011864816445
SVR test mse: 59.79989098398971
```

Regularised linear regression

```
In [24]: lin_model = Lasso(random_state=2909)
lin_model.fit(scaler.transform(X_train[training_vars]), y_train)

pred = lin_model.predict(scaler.transform(X_train[training_vars]))
print('linear train mse: {}'.format(mean_squared_error(y_train, pred
)))
pred = lin_model.predict(scaler.transform(X_test[training_vars]))
print('linear test mse: {}'.format(mean_squared_error(y_test, pred))
)
```

```
linear train mse: 30.79952937572478
linear test mse: 53.28622729247063
```

Submission to DataDriven

In [ ]:
```
pred_ls = []  #combine xgboost and random forest results
for model in [xgb_model, rf_model]:
    pred_ls.append(pd.Series(model.predict(submission[training_vars]
)))

#pred = SVR_model.predict(scaler.transform(submission[training_vars]
))
#pred_ls.append(pd.Series(pred))

#pred = lin_model.predict(scaler.transform(submission[training_vars]
))
#pred_ls.append(pd.Series(pred))

final_pred = pd.concat(pred_ls, axis=1).mean(axis=1)
```

In [ ]:
```
temp = pd.concat([submission.row_id, final_pred], axis=1)
temp.columns = ['row_id', 'prevalence_of_undernourishment']
temp.head()
```

In [ ]:
```
temp.to_csv('resultxgrf.csv', index=False)
```

In [ ]:
```
#For XGBoost
pred_ls5 = []
for model in [xgb_model]:
    pred_ls5.append(pd.Series(model.predict(submission[training_vars
])))
final_pred = pd.concat(pred_ls5, axis=1).mean(axis=1)
```

In [ ]:
```
temp5 = pd.concat([submission.row_id, final_pred], axis=1)
temp5.columns = ['row_id', 'prevalence_of_undernourishment']
temp5.head()
```

In [ ]:
```
#pd.pandas.set_option('display.max_rows', None)
#display(temp)
```

In [ ]:
```
temp5.to_csv('resultxg.csv', index=False)
```

In [ ]:
```
#For Random Forest
pred_ls2 = []
for model in [rf_model]:
    pred_ls2.append(pd.Series(model.predict(submission[training_vars
])))
final_pred = pd.concat(pred_ls2, axis=1).mean(axis=1)
```

In [ ]:
```
temp2 = pd.concat([submission.row_id, final_pred], axis=1)
temp2.columns = ['row_id', 'prevalence_of_undernourishment']
temp2.head()
```

In [ ]:
```
temp2.to_csv('resultrf.csv', index=False)
```

In [ ]:
```
#For Support Vector Machine
pred_ls3 = []
for model in [SVR_model]:
    pred_ls3.append(pd.Series(model.predict(submission[training_vars
])))
final_pred = pd.concat(pred_ls3, axis=1).mean(axis=1)
```

In [ ]:
```
temp3 = pd.concat([submission.row_id, final_pred], axis=1)
temp3.columns = ['row_id', 'prevalence_of_undernourishment']
temp3.head()
```

In [ ]:
```
temp3.to_csv('resultsvm.csv', index=False)
```

In [ ]:
```
#For Linear regression
pred_ls4 = []
for model in [lin_model]:
    pred_ls4.append(pd.Series(model.predict(submission[training_vars
])))
final_pred = pd.concat(pred_ls4, axis=1).mean(axis=1)
```

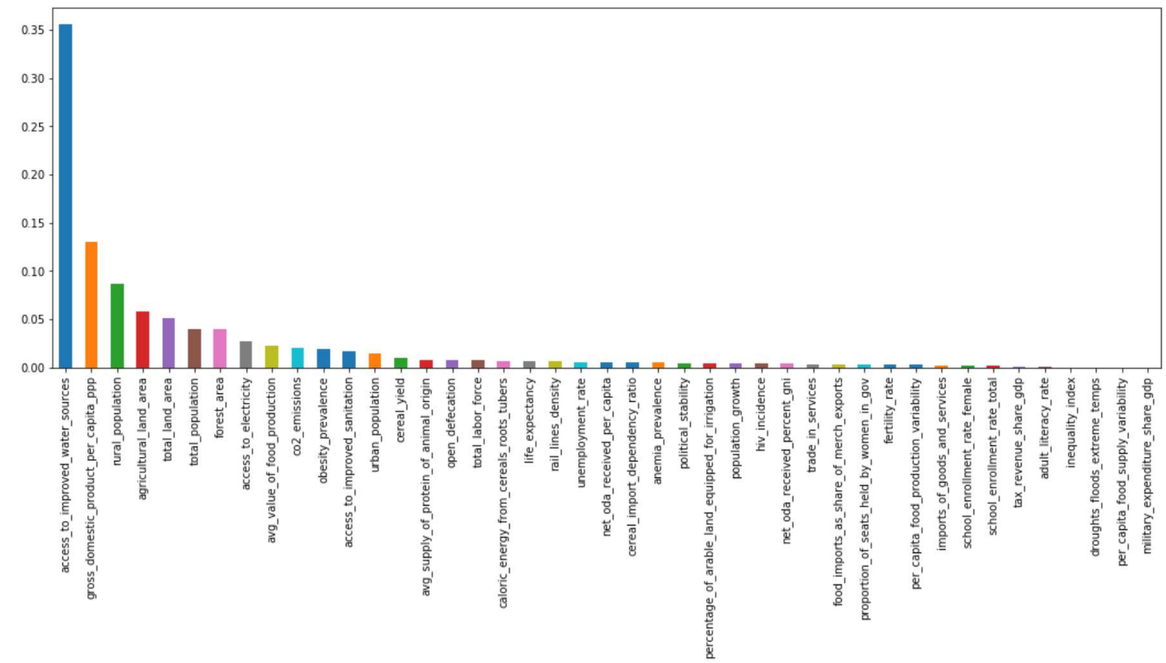
In [ ]:
```
temp4 = pd.concat([submission.row_id, final_pred], axis=1)
temp4.columns = ['row_id', 'prevalence_of_undernourishment']
temp4.head()
```

In [ ]:
```
temp4.to_csv('resultslin.csv', index=False)
```

Feature importance

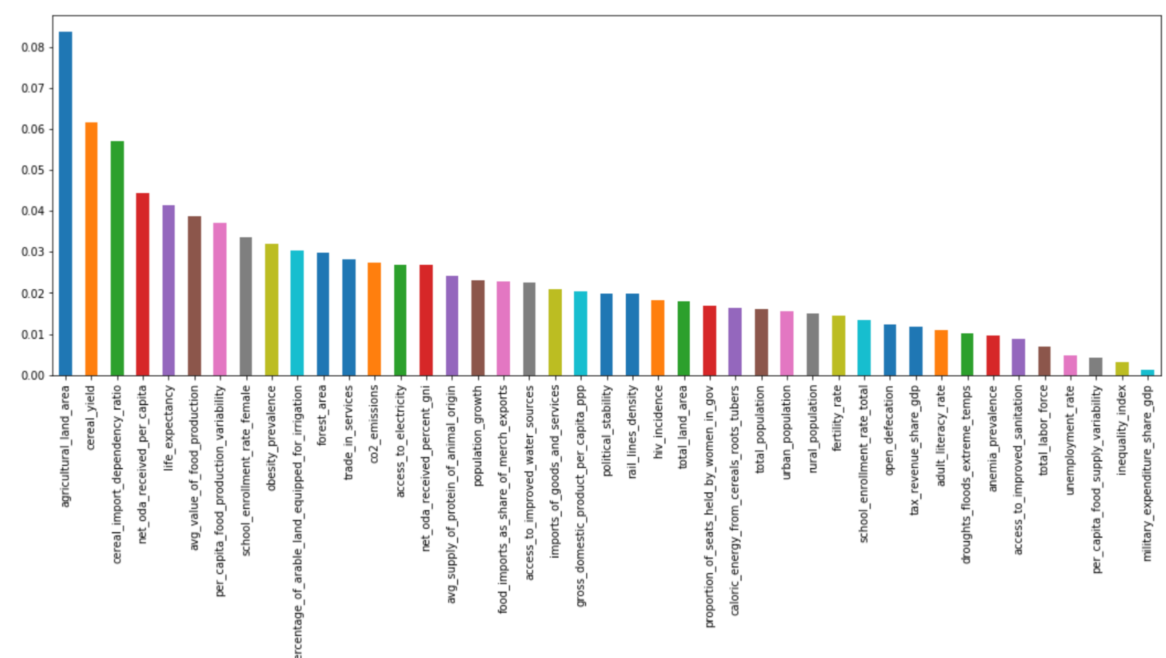
In [25]:
```
importance = pd.Series(rf_model.feature_importances_)
importance.index = training_vars
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(18,6))
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0xd6619ea20>



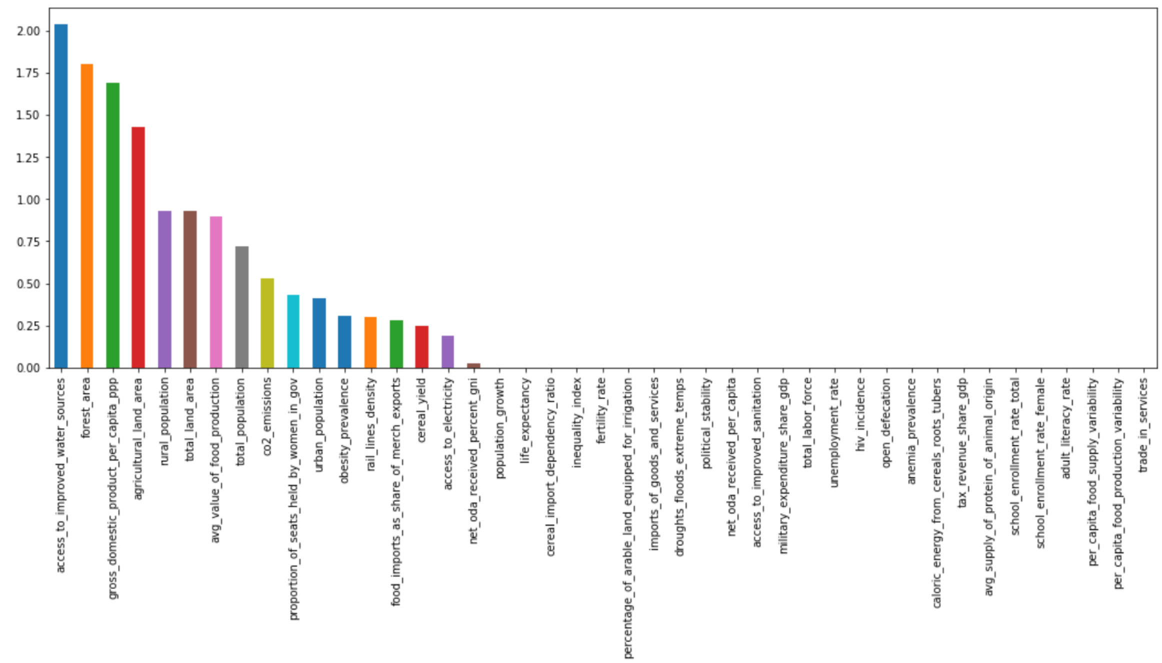
In [26]:
```
importance = pd.Series(xgb_model.feature_importances_)
importance.index = training_vars
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(18,6))
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0xd66a66cc0>



In [27]:
```
importance = pd.Series(np.abs(lin_model.coef_.ravel()))
importance.index = training_vars
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(18,6))
```

Out[27]: `<matplotlib.axes._subplots.AxesSubplot at 0xd66b1dc88>`



In [28]:
```
importance = pd.Series(np.abs(SVR_model.coef_.ravel()))
importance.index = training_vars
importance.sort_values(inplace=True, ascending=False)
importance.plot.bar(figsize=(18,6))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-28-26ab1fde4b19> in <module>
----> 1 importance = pd.Series(np.abs(SVR_model.coef_.ravel()))
      2 importance.index = training_vars
      3 importance.sort_values(inplace=True, ascending=False)
      4 importance.plot.bar(figsize=(18,6))

~\Anaconda3\lib\site-packages\sklearn\svm\base.py in coef_(self)
    482     def coef_(self):
    483         if self.kernel != 'linear':
--> 484             raise AttributeError('coef_ is only available when using a '
    485                                  'linear kernel')
    486 

AttributeError: coef_ is only available when using a linear kernel
```