In [2]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input direct

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 5GB to the current directory (/kaggle/working/) that gets preserved as output when you cre
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/usaids-model-future-contraceptive-use/Train (2).csv
/kaggle/input/usaids-model-future-contraceptive-use/contraceptive_case_data_annual.csv
/kaggle/input/usaids-model-future-contraceptive-use/data_dictionary.csv
/kaggle/input/usaids-model-future-contraceptive-use/product.csv
/kaggle/input/usaids-model-future-contraceptive-use/Recommended_Supplementary_Data_Sources.docx
/kaggle/input/usaids-model-future-contraceptive-use/monthly_cases.csv
/kaggle/input/usaids-model-future-contraceptive-use/service_delivery_site_data.csv
/kaggle/input/usaids-model-future-contraceptive-use/annual_cases.csv
/kaggle/input/usaids-model-future-contraceptive-use/Table_of_Contents.xlsx
/kaggle/input/usaids-model-future-contraceptive-use/SampleSubmission (2).csv
/kaggle/input/contra-boost/Contra_service_delivery_site_data.csv
```

In [ ]:
```
%matplotlib inline

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
import seaborn as sns
import matplotlib.pyplot as plt
import time


#from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split,GridSearchCV
import matplotlib.pylab as plt
from sklearn.tree import export_graphviz
from sklearn import tree

from xgboost import XGBRegressor
from sklearn.model_selection import RandomizedSearchCV
```

In [ ]:
```
train=pd.read_csv('../input/usaids-model-future-contraceptive-use/Train (2).csv')
sub=pd.read_csv('../input/usaids-model-future-contraceptive-use/SampleSubmission (2).csv')
sd=pd.read_csv('../input/contra-boost/Contra_service_delivery_site_data.csv')
```

In [ ]:
```
train.columns
```

In [ ]:
```
train.drop(columns={'stock_initial', 'stock_received','stock_adjustment',
                    'stock_end', 'average_monthly_consumption',
                    'stock_stockout_days', 'stock_ordered'},inplace=True)
```

In [7]:
```
test=pd.DataFrame(sub.ID.str.split('X',3).tolist(), columns = ['year','month','site_code','product_code'])
for col in test.columns:
    test[col] = test[col].str.strip()
column=['year','month']
for col in column:
    test[col]=test[col].astype(int)
test=pd.merge(test,sd,on='site_code',how='left')
test.rename(columns={'site_region':'region','site_district':'district'},inplace=True)
test=test[['year', 'month', 'region','district','site_code', 'product_code']]
```

In [8]:
```
train.head(2).append(test.head(2))
```

Out[8]:

	year	month	region	district	site_code	product_code	stock_distributed
0	2019	1	INDENIE-DJUABLIN	ABENGOUROU	C4001	AS27134	21.0
1	2019	1	INDENIE-DJUABLIN	ABENGOUROU	C4001	AS27132	3.0
0	2019	7	INDENIE-DJUABLIN	ABENGOUROU	C4001	AS21126	NaN
1	2019	7	INDENIE-DJUABLIN	ABENGOUROU	C4001	AS27134	NaN

In [9]:
```
train['train_or_test']='train'
test['train_or_test']='test'
df=pd.concat([train,test])
```

In [10]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

for col in ['region','district','site_code','product_code']:
    df[col]=  df[col].astype('str')
    df[col]= le.fit_transform(df[col])
```

In [11]:
```
train=df.loc[df.train_or_test.isin(['train'])]
test=df.loc[df.train_or_test.isin(['test'])]
train.drop(columns={'train_or_test'},axis=1,inplace=True)
test.drop(columns={'train_or_test'},axis=1,inplace=True)
```

```
/opt/conda/lib/python3.7/site-packages/pandas/core/frame.py:3997: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  errors=errors,
```

In [12]:
```
train.head()
```

Out[12]:

	year	month	region	district	site_code	product_code	stock_distributed
0	2019	1	11	0	119	5	21.0
1	2019	1	11	0	119	3	3.0
2	2019	1	11	0	119	2	22.0
3	2019	1	11	0	119	6	0.0
4	2019	1	11	0	119	7	2.0

In [13]:
```
train['product_code'] = train['product_code'].astype('category')
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [14]:
```
test['product_code'] = test['product_code'].astype('category')
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  """Entry point for launching an IPython kernel.
```

In [15]:
```
train =pd.get_dummies(train, drop_first = True)
```

In [16]:
```
train.columns
```

Out[16]:
```
Index(['year', 'month', 'region', 'district', 'site_code', 'stock_distributed',
       'product_code_1', 'product_code_2', 'product_code_3', 'product_code_4',
       'product_code_5', 'product_code_6', 'product_code_7', 'product_code_8',
       'product_code_9', 'product_code_10'],
      dtype='object')
```

In [17]:
```
test =pd.get_dummies(test, drop_first = True)
```

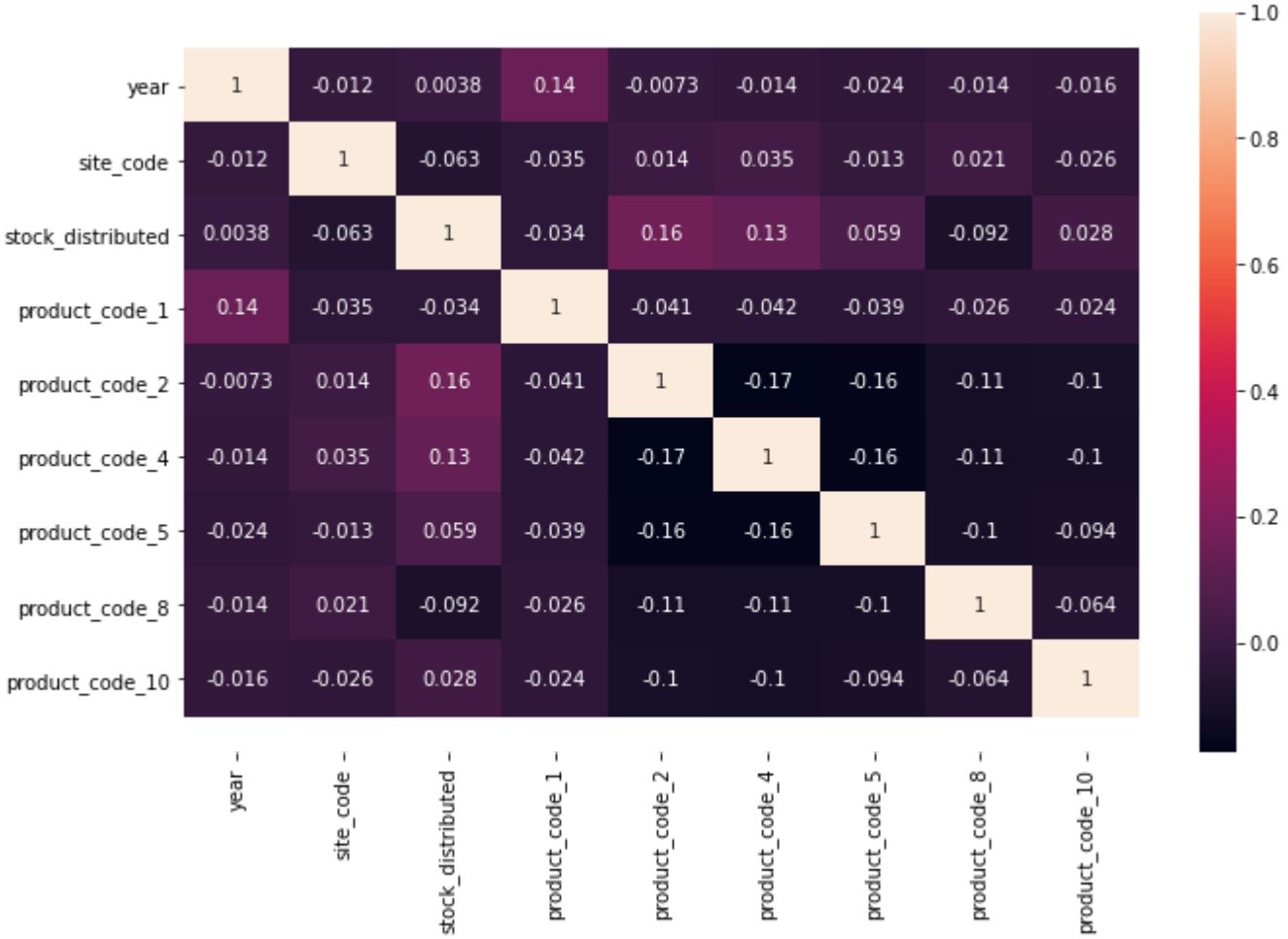
In [18]:
```
len(test),len(sub)
```

Out[18]:
```
(3089, 3089)
```

In [19]:
```
train.drop(columns = ['month', 'district', 'region', 'product_code_6', 'product_code_3', 'product_code_7', 'proc
test.drop(columns = ['month', 'district', 'region', 'product_code_6', 'product_code_3', 'product_code_7', 'produ
```

In [20]:
```
corr = train.corr()
fig, ax = plt.subplots()
fig.set_size_inches(11, 7)
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns,annot = True, ax = ax)
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[20]: (9.5, -0.5)



In [21]:
```
train['stock_distributed'] = np.where(train['stock_distributed'] == 0, train['stock_distributed'].median(), trai
```

In [22]:
```
X=train.drop(columns={'stock_distributed'})
y=train.loc[:,['stock_distributed']]
del test['stock_distributed']


train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.4, random_state=15)
```

In [23]:
```
#Define the model
my_model = XGBRegressor()
```

In [24]:
```
##Hyper Parameter Optimization
booster = ['gbtree', 'gblinear']
base_score = [0.25, 0.5, 0.75, 1]
eta = [0.01, 0.015, 0.025, 0.05, 0.1]
gamma = [0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.3, 0.5,0.7,0.9,1.0]
n_estimators = [100, 500, 900, 1100, 1500]
max_depth = [2, 3, 5, 10, 15]
min_child_weight = [1,3,5,7]
subsample = [0.6, 0.7, 0.8, 0.9, 1.0]
colsample_bytree = [0.6, 0.7, 0.8, 0.9, 1.0]
reg_lambda = [0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 1.0]
reg_alpha = [0, 0.1, 0.5, 1.0, 1.5]
booster = ['gbtree', 'gblinear']
learning_rate = [0.05, 0.1, 0.15, 0.20]
min_child_weight = [1, 2, 3, 4]

#Define the grid of hyperparameters to search
hyperparameter_grid = {
    'n_estimators' : n_estimators,
    'max_depth': max_depth,
    'learning_rate': learning_rate,
    'min_child_weight': min_child_weight,
    'booster': booster,
    'base_score': base_score
     }
```

In [25]:
```
#Set up the random search with 4-fold cross validation
random_cv = RandomizedSearchCV(estimator=my_model, param_distributions = hyperparameter_grid,
                               cv = 5, n_iter = 50,
                               scoring = 'neg_mean_absolute_error', n_jobs = 4,
                               verbose = 5,
                               return_train_score = True,
                               random_state = 42)
```

In [26]:
```
#fit the model
random_cv.fit(train_X, train_y)
```

```
Fitting 5 folds for each of 50 candidates, totalling 250 fits

[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  10 tasks      | elapsed:   26.2s
/opt/conda/lib/python3.7/site-packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker st
opped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a mem
ory leak.
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=4)]: Done  64 tasks      | elapsed:  4.7min
[Parallel(n_jobs=4)]: Done 154 tasks      | elapsed:  7.9min
[Parallel(n_jobs=4)]: Done 250 out of 250 | elapsed: 11.9min finished
```

Out[26]:
```
RandomizedSearchCV(cv=5,
                   estimator=XGBRegressor(base_score=None, booster=None,
                                          colsample_bylevel=None,
                                          colsample_bynode=None,
                                          colsample_bytree=None, gamma=None,
                                          gpu_id=None, importance_type='gain',
                                          interaction_constraints=None,
                                          learning_rate=None,
                                          max_delta_step=None, max_depth=None,
                                          min_child_weight=None, missing=nan,
                                          monotone_constraints=None,
                                          n_estimators=100, n...
                                          validate_parameters=None,
                                          verbosity=None),
                   n_iter=50, n_jobs=4,
                   param_distributions={'base_score': [0.25, 0.5, 0.75, 1],
                                        'booster': ['gbtree', 'gblinear'],
                                        'learning_rate': [0.05, 0.1, 0.15, 0.2],
                                        'max_depth': [2, 3, 5, 10, 15],
                                        'min_child_weight': [1, 2, 3, 4],
                                        'n_estimators': [100, 500, 900, 1100,
                                                         1500]},
                   random_state=42, return_train_score=True,
                   scoring='neg_mean_absolute_error', verbose=5)
```

In [27]:
```
random_cv.best_estimator_
```

Out[27]:
```
XGBRegressor(base_score=1, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=15,
             min_child_weight=1, missing=nan, monotone_constraints='()',
             n_estimators=900, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [28]:
```
random_cv.best_params_
```

Out[28]:
```
{'n_estimators': 900,
 'min_child_weight': 1,
 'max_depth': 15,
 'learning_rate': 0.1,
 'booster': 'gbtree',
 'base_score': 1}
```

In [29]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'booster': [random_cv.best_params_['booster']],
    'max_depth': [random_cv.best_params_['max_depth']],
    'learning_rate': [random_cv.best_params_['learning_rate']],
    'min_child_weight': [random_cv.best_params_['min_child_weight'],
                         random_cv.best_params_['min_child_weight']+2,
                         random_cv.best_params_['min_child_weight'] + 4],
    'base_score': [random_cv.best_params_['base_score'] + 2,
                          random_cv.best_params_['base_score'] +1,
                          random_cv.best_params_['base_score'],
                          random_cv.best_params_['base_score'] +3,
                          random_cv.best_params_['base_score'] + 5],
    'n_estimators': [random_cv.best_params_['n_estimators'] - 200, random_cv.best_params_['n_estimators'] - 100,
                     random_cv.best_params_['n_estimators'],
                     random_cv.best_params_['n_estimators'] + 100, random_cv.best_params_['n_estimators'] + 200]
}

print(param_grid)
```

```
{'booster': ['gbtree'], 'max_depth': [15], 'learning_rate': [0.1], 'min_child_weight': [1, 3, 5], 'base_scor
e': [3, 2, 1, 4, 6], 'n_estimators': [700, 800, 900, 1000, 1100]}
```

In [30]:
```
#### Fit the grid_search to the data
my_model = XGBRegressor()
grid_search=GridSearchCV(estimator=my_model,param_grid=param_grid,cv=10,n_jobs=-1,verbose=2)
grid_search.fit(train_X,train_y)
```

```
Fitting 10 folds for each of 75 candidates, totalling 750 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  2.9min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed: 23.2min
[Parallel(n_jobs=-1)]: Done 357 tasks      | elapsed: 50.7min
[Parallel(n_jobs=-1)]: Done 640 tasks      | elapsed: 94.4min
[Parallel(n_jobs=-1)]: Done 750 out of 750 | elapsed: 113.9min finished
```

Out[30]:
```
GridSearchCV(cv=10,
             estimator=XGBRegressor(base_score=None, booster=None,
                                    colsample_bylevel=None,
                                    colsample_bynode=None,
                                    colsample_bytree=None, gamma=None,
                                    gpu_id=None, importance_type='gain',
                                    interaction_constraints=None,
                                    learning_rate=None, max_delta_step=None,
                                    max_depth=None, min_child_weight=None,
                                    missing=nan, monotone_constraints=None,
                                    n_estimators=100, n_jobs...
                                    num_parallel_tree=None, random_state=None,
                                    reg_alpha=None, reg_lambda=None,
                                    scale_pos_weight=None, subsample=None,
                                    tree_method=None, validate_parameters=None,
                                    verbosity=None),
             n_jobs=-1,
             param_grid={'base_score': [3, 2, 1, 4, 6], 'booster': ['gbtree'],
                         'learning_rate': [0.1], 'max_depth': [15],
                         'min_child_weight': [1, 3, 5],
                         'n_estimators': [700, 800, 900, 1000, 1100]},
             verbose=2)
```

In [31]:
```
grid_search.best_estimator_
```

Out[31]:
```
XGBRegressor(base_score=6, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=15,
             min_child_weight=5, missing=nan, monotone_constraints='()',
             n_estimators=700, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [33]:
```
best_grid=grid_search.best_estimator_
```

In [34]:
```
best_grid
```

Out[34]:
```
XGBRegressor(base_score=6, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
             importance_type='gain', interaction_constraints='',
             learning_rate=0.1, max_delta_step=0, max_depth=15,
             min_child_weight=5, missing=nan, monotone_constraints='()',
             n_estimators=700, n_jobs=0, num_parallel_tree=1, random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
             tree_method='exact', validate_parameters=1, verbosity=None)
```

In [35]:
```
#my_model.fit(train_X, train_y)
pred = best_grid.predict(test)
```

In [36]:
```
sub['prediction']=np.abs(pred)
```

In [37]:
```
sub['prediction'] = sub['prediction'].round()
```

In [38]:
```
sub.to_csv('XGBoost19.csv',index=False)
```