Regression with a Flood Prediction Dataset (Playground Series - Season 4, Episode 5)

Descriptions

The dataset for this competition (both train and test) was generated from a deep learning model trained on the Flood Prediction Factors dataset. Feature distributions are close to, but not exactly the same, as the original.

Files train.csv - the training dataset; FloodProbability is the target
test.csv - the test dataset; your objective is to predict the FloodProbability for each row
sample_submission.csv - a sample submission file in the correct format

Objectives

The goal of this competition is to predict the probability of a region flooding based on various factors.

About Author

Name: Shuvendu Pritam Das

Email Id: shuvendupritamdas181@gmail.com

LinkedIn Id: https://www.linkedin.com/in/shuvendupritamdas/

1. Importing Libraries and Data sets

1.1 Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn
```

```
import plotly.express as px
import warnings
warnings.filterwarnings('ignore')
```

1.2 Data Sets

In [2]:
```
df_train = pd.read_csv(r"C:\Users\shuve\Desktop\ML((GFG)\Kaggle Comp\Fload\train.csv")
df_test = pd.read_csv(r"C:\Users\shuve\Desktop\ML((GFG)\Kaggle Comp\Fload\test.csv")
```

In [3]:
```
df_train.head(3)
```

Out[3]:

	id	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPracti
0	0	5	8	5	8	6	4	4	3	
1	1	6	7	4	4	8	8	3	5	
2	2	6	5	6	7	3	7	1	5	

3 rows × 22 columns

In [4]:
```
df_test.head(3)
```

Out[4]:

	id	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	Agricultural
0	1117957	4	6	3	5	6	7	8	7	
1	1117958	4	4	2	9	5	5	4	7	
2	1117959	1	3	6	5	7	2	4	6	

3 rows × 21 columns

In [5]:
```
df_train.columns
```

Out[5]:
```
Index(['id', 'MonsoonIntensity', 'TopographyDrainage', 'RiverManagement',
       'Deforestation', 'Urbanization', 'ClimateChange', 'DamsQuality',
       'Siltation', 'AgriculturalPractices', 'Encroachments',
       'IneffectiveDisasterPreparedness', 'DrainageSystems',
       'CoastalVulnerability', 'Landslides', 'Watersheds',
       'DeterioratingInfrastructure', 'PopulationScore', 'WetlandLoss',
       'InadequatePlanning', 'PoliticalFactors', 'FloodProbability'],
      dtype='object')
```

In [6]:
```
df_test.columns
```

Out[6]:
```
Index(['id', 'MonsoonIntensity', 'TopographyDrainage', 'RiverManagement',
       'Deforestation', 'Urbanization', 'ClimateChange', 'DamsQuality',
       'Siltation', 'AgriculturalPractices', 'Encroachments',
       'IneffectiveDisasterPreparedness', 'DrainageSystems',
       'CoastalVulnerability', 'Landslides', 'Watersheds',
       'DeterioratingInfrastructure', 'PopulationScore', 'WetlandLoss',
       'InadequatePlanning', 'PoliticalFactors'],
      dtype='object')
```

1.3 Defining Features and Targets

Features:

1. 'id'
2. 'MonsoonIntensity'
3. 'TopographyDrainage'
4. 'RiverManagement'
5. 'Deforestation'
6. 'Urbanization'
7. 'ClimateChange'
8. 'DamsQuality'
9. 'Siltation'
10. 'AgriculturalPractices'
11. 'Encroachments'
12. 'IneffectiveDisasterPreparedness'
13. 'DrainageSystems'
14. 'CoastalVulnerability'

15. 'Landslides'
16. 'Watersheds'
17. 'DeterioratingInfrastructure'
18. 'PopulationScore'
19. 'WetlandLoss'
20. 'InadequatePlanning'
21. 'PoliticalFactors'
    #### **Target**:
22. 'FloodProbability'

2.Data Exploration

2.1 Dimensions

In [7]:
```
df_train.shape,df_test.shape
```

Out[7]:
```
((1117957, 22), (745305, 21))
```

2.2 Statistical Summary

In [8]:
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1117957 entries, 0 to 1117956
Data columns (total 22 columns):
 #   Column                           Non-Null Count    Dtype  
---  ------                           --------------    -----  
 0   id                               1117957 non-null  int64  
 1   MonsoonIntensity                 1117957 non-null  int64  
 2   TopographyDrainage               1117957 non-null  int64  
 3   RiverManagement                  1117957 non-null  int64  
 4   Deforestation                    1117957 non-null  int64  
 5   Urbanization                     1117957 non-null  int64  
 6   ClimateChange                    1117957 non-null  int64  
 7   DamsQuality                      1117957 non-null  int64  
 8   Siltation                        1117957 non-null  int64  
 9   AgriculturalPractices            1117957 non-null  int64  
 10  Encroachments                    1117957 non-null  int64  
 11  IneffectiveDisasterPreparedness  1117957 non-null  int64  
 12  DrainageSystems                  1117957 non-null  int64  
 13  CoastalVulnerability             1117957 non-null  int64  
 14  Landslides                       1117957 non-null  int64  
 15  Watersheds                       1117957 non-null  int64  
 16  DeterioratingInfrastructure      1117957 non-null  int64  
 17  PopulationScore                  1117957 non-null  int64  
 18  WetlandLoss                      1117957 non-null  int64  
 19  InadequatePlanning               1117957 non-null  int64  
 20  PoliticalFactors                 1117957 non-null  int64  
 21  FloodProbability                 1117957 non-null  float64
dtypes: float64(1), int64(21)
memory usage: 187.6 MB
```

2.3 Dropping Id Column

In [9]:

```
df_train.drop(columns= ["id"], inplace=True)
df_train.head(2)
```

Out[9]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices
0	5	8	5	8	6	4	4	3	3
1	6	7	4	4	8	8	3	5	4

2 rows × 21 columns

2.4 Check for Duplicacy

In [10]:
```
df_train.duplicated().sum()
```

Out[10]:
```
0
```

Remark: No duplicacy found

2.5 Check for Null values

In [11]:
```
df_train.isnull().sum()
```

Out[11]:
```
MonsoonIntensity                   0
TopographyDrainage                 0
RiverManagement                    0
Deforestation                      0
Urbanization                       0
ClimateChange                      0
DamsQuality                        0
Siltation                          0
AgriculturalPractices              0
Encroachments                      0
IneffectiveDisasterPreparedness    0
DrainageSystems                    0
CoastalVulnerability               0
Landslides                         0
Watersheds                         0
DeterioratingInfrastructure        0
PopulationScore                    0
WetlandLoss                        0
InadequatePlanning                 0
PoliticalFactors                   0
FloodProbability                   0
dtype: int64
```

Remark: No Null value Found

2.6 Descriptive Analytics

In [12]:
```
df_train.describe()
```

Out[12]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	Agricultural
count	1.117957e+06	1.117957e+06	1.117957e+06	1.117957e+06	1.117957e+06	1.117957e+06	1.117957e+06	1.117957e+06	1.11
mean	4.921450e+00	4.926671e+00	4.955322e+00	4.942240e+00	4.942517e+00	4.934093e+00	4.955878e+00	4.927791e+00	4.94
std	2.056387e+00	2.093879e+00	2.072186e+00	2.051689e+00	2.083391e+00	2.057742e+00	2.083063e+00	2.065992e+00	2.06
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00
25%	3.000000e+00	3.000000e+00	4.000000e+00	4.000000e+00	3.000000e+00	3.000000e+00	4.000000e+00	3.000000e+00	3.00
50%	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.00
75%	6.000000e+00	6.000000e+00	6.000000e+00	6.000000e+00	6.000000e+00	6.000000e+00	6.000000e+00	6.000000e+00	6.00
max	1.600000e+01	1.800000e+01	1.600000e+01	1.700000e+01	1.700000e+01	1.700000e+01	1.600000e+01	1.600000e+01	1.60

8 rows × 21 columns

3. EDA

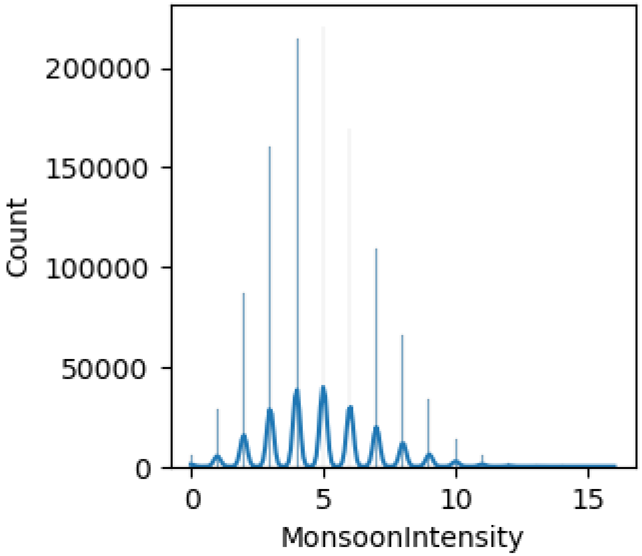
3.1 Univariate Analysis

3.1.1 Distributions

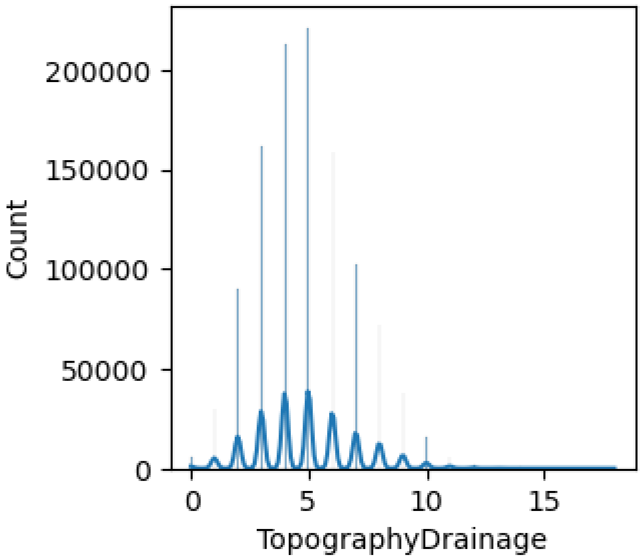
In [13]:

```
for i in df_train.columns:
    plt.figure(figsize = (3,3))
    print(sns.histplot(df_train[i],kde = True))
    plt.show()
```

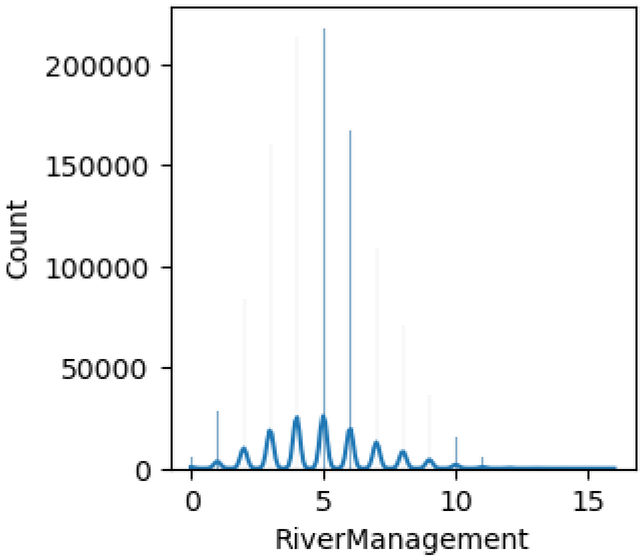
```
Axes(0.125,0.11;0.775x0.77)
```



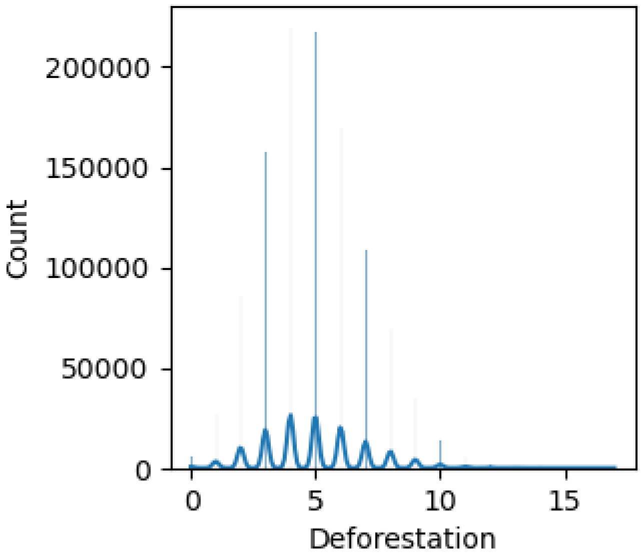

```
Axes(0.125,0.11;0.775x0.77)
```



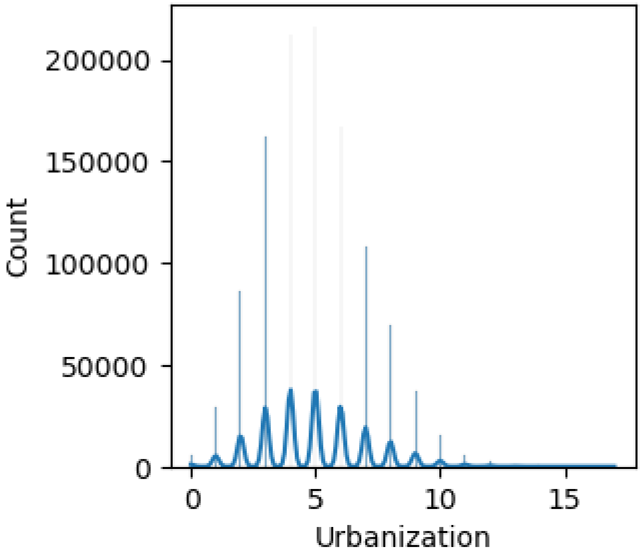

```
Axes(0.125,0.11;0.775x0.77)
```



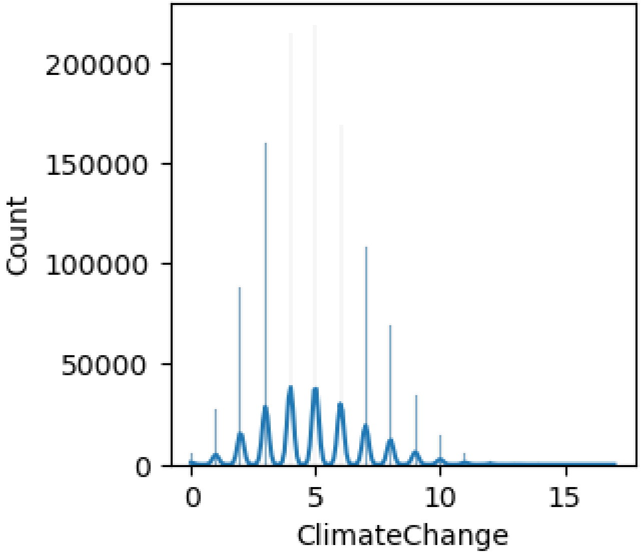

```
Axes(0.125,0.11;0.775x0.77)
```



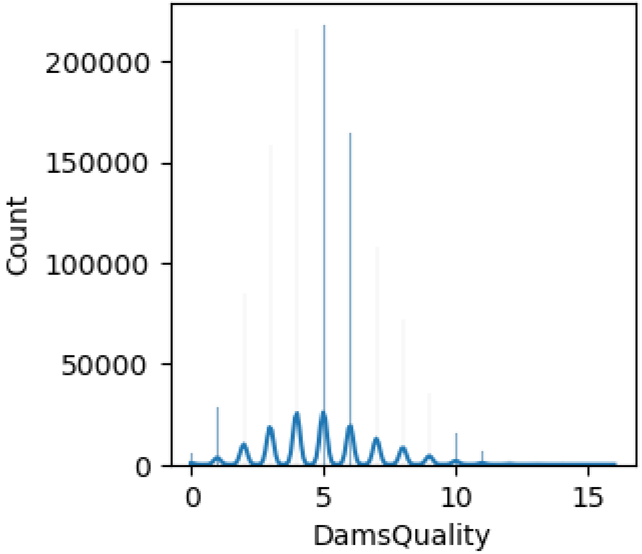

```
Axes(0.125,0.11;0.775x0.77)
```



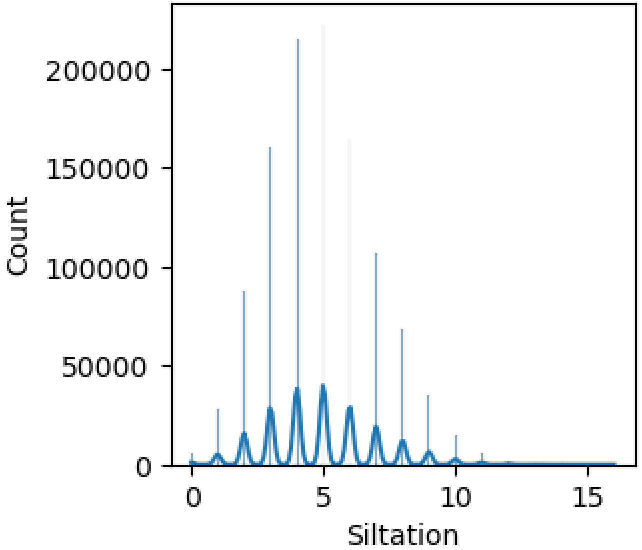

```
Axes(0.125,0.11;0.775x0.77)
```



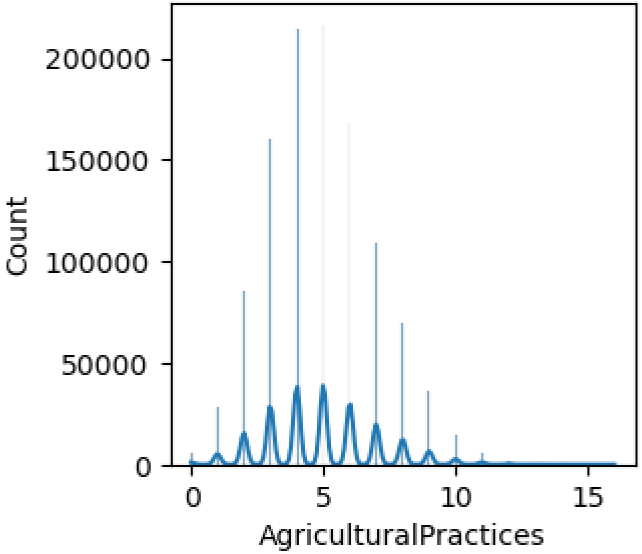

```
Axes(0.125,0.11;0.775x0.77)
```



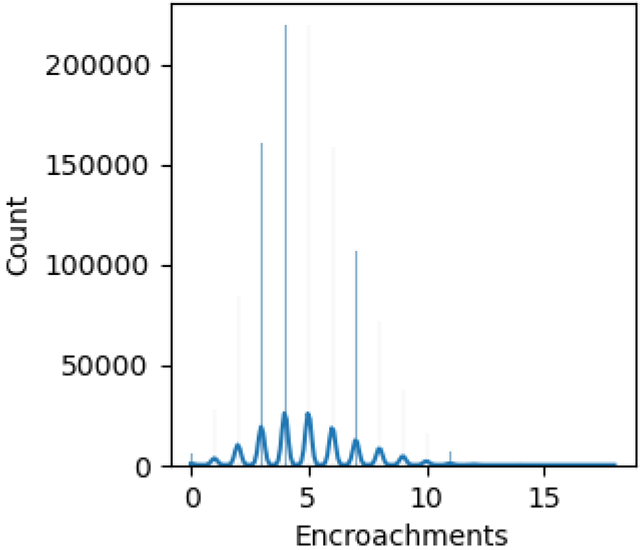

```
Axes(0.125,0.11;0.775x0.77)
```



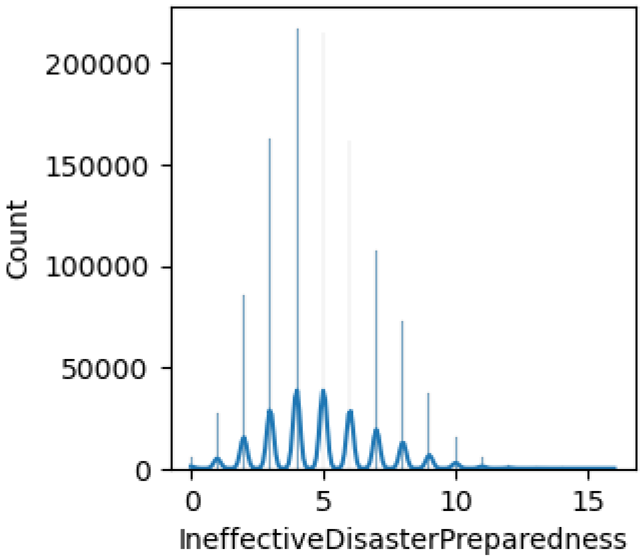

```
Axes(0.125,0.11;0.775x0.77)
```



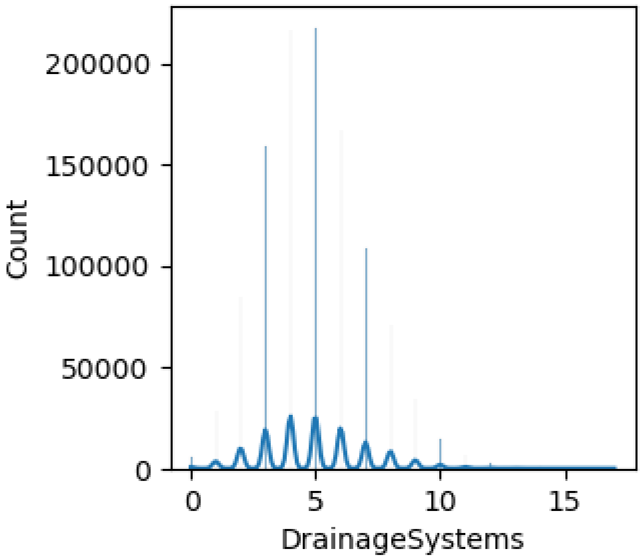

```
Axes(0.125,0.11;0.775x0.77)
```



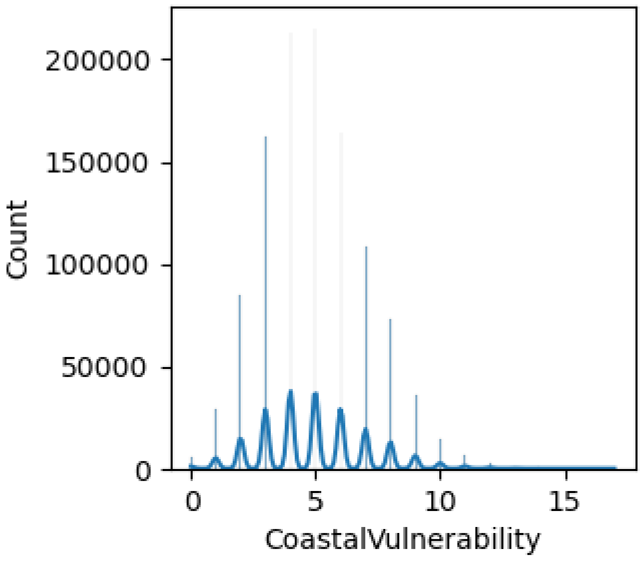

```
Axes(0.125,0.11;0.775x0.77)
```



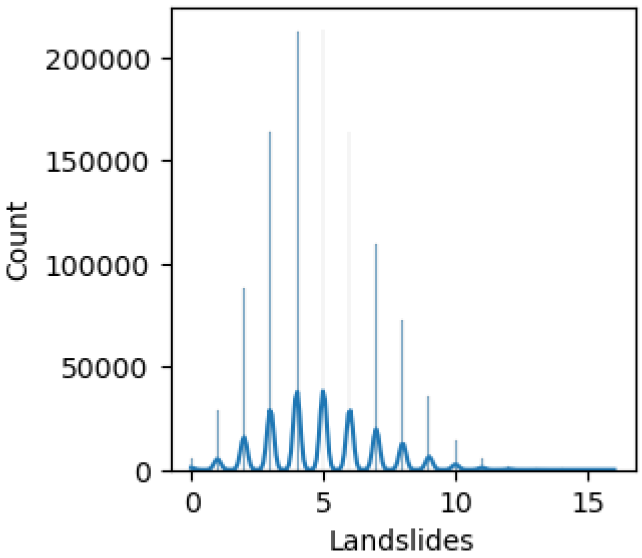

```
Axes(0.125,0.11;0.775x0.77)
```



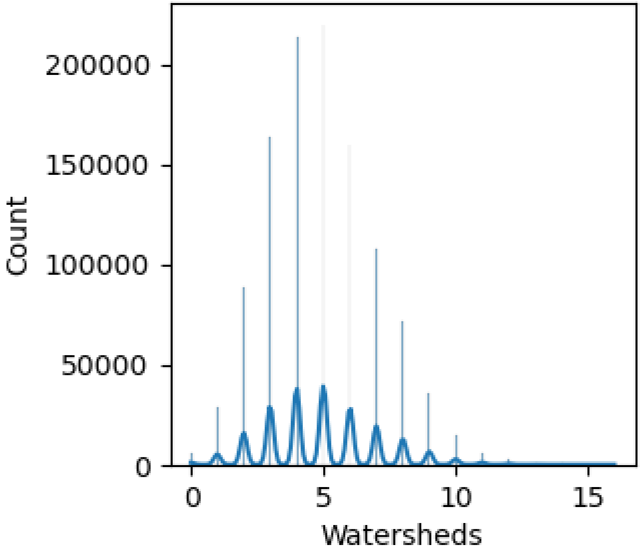

```
Axes(0.125,0.11;0.775x0.77)
```



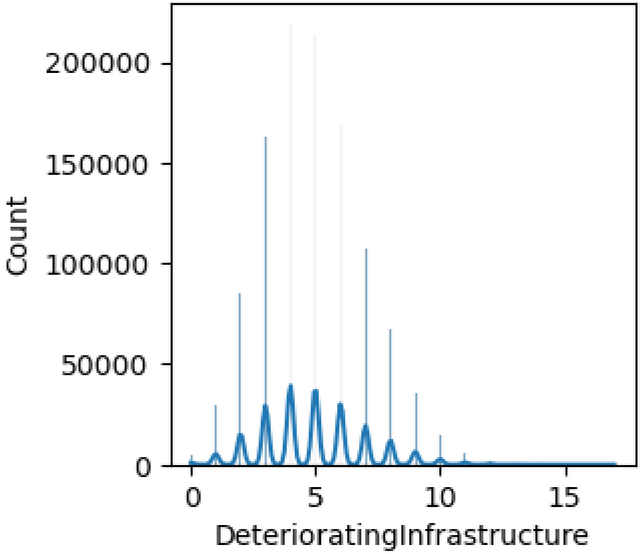

```
Axes(0.125,0.11;0.775x0.77)
```



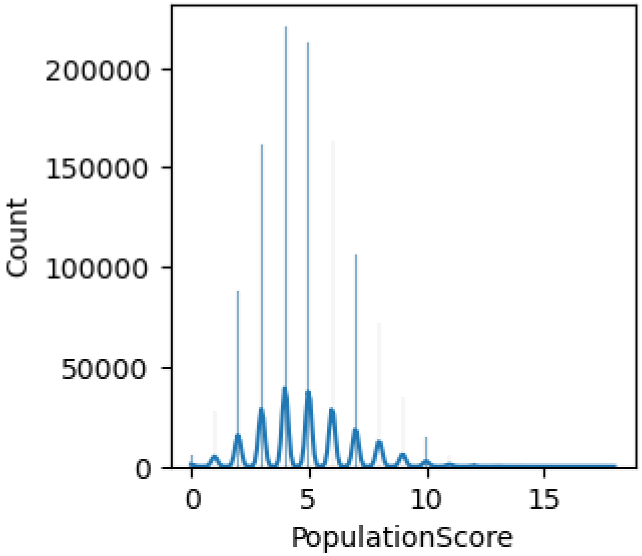

```
Axes(0.125,0.11;0.775x0.77)
```



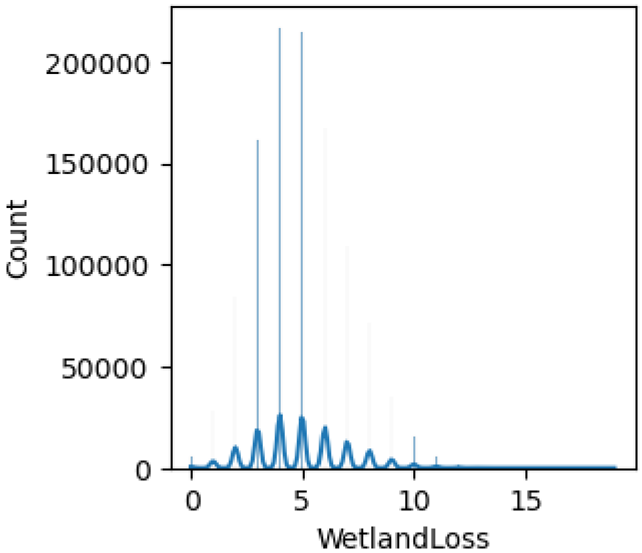

```
Axes(0.125,0.11;0.775x0.77)
```



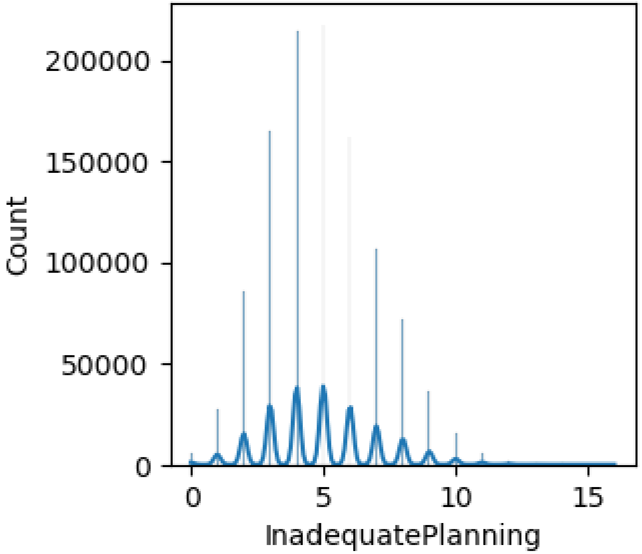

```
Axes(0.125,0.11;0.775x0.77)
```



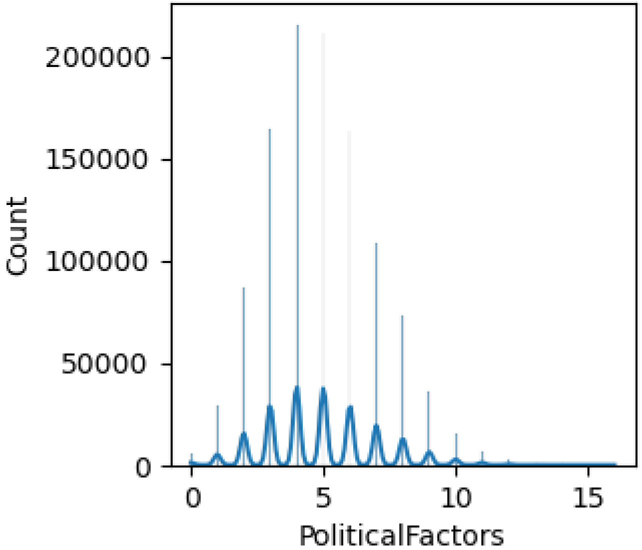

```
Axes(0.125,0.11;0.775x0.77)
```



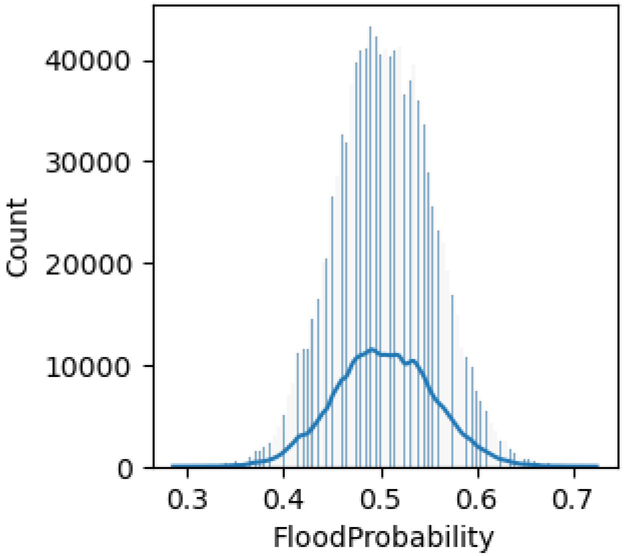

```
Axes(0.125,0.11;0.775x0.77)
```




```
Axes(0.125,0.11;0.775x0.77)
```



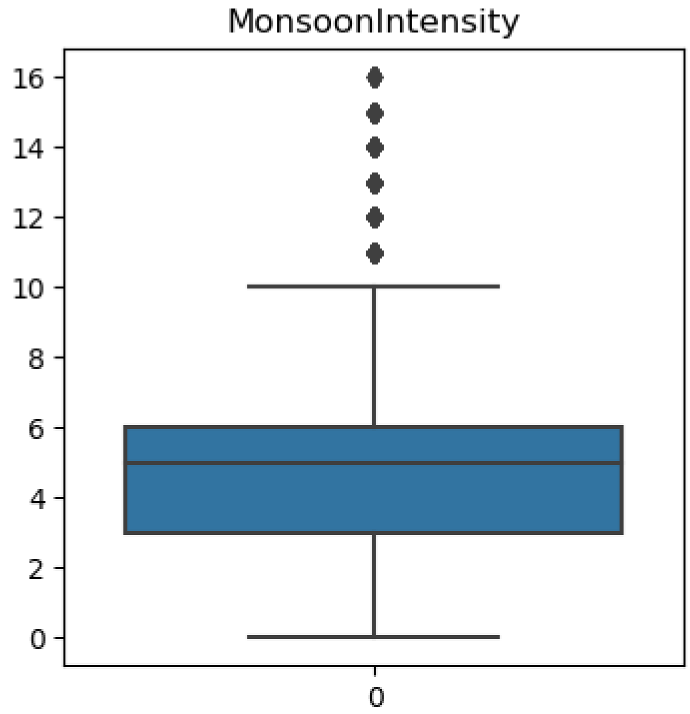

```
Axes(0.125,0.11;0.775x0.77)
```



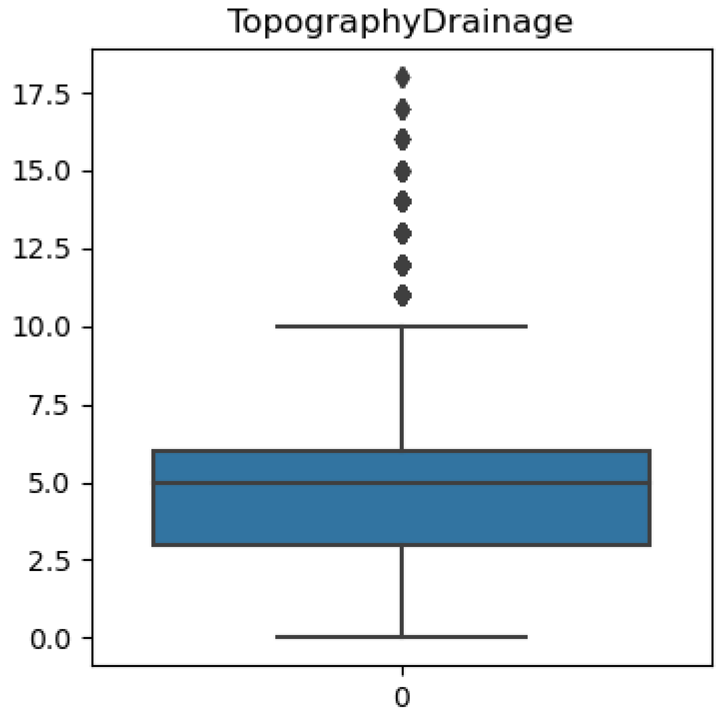
3.1.2 Box-Plot

In [14]:
```
for i in df_train.columns:
    plt.figure(figsize = (3,3))
    print(sns.boxplot(df_train[i]), end = " ")
    plt.title(i)
    plt.show()
```

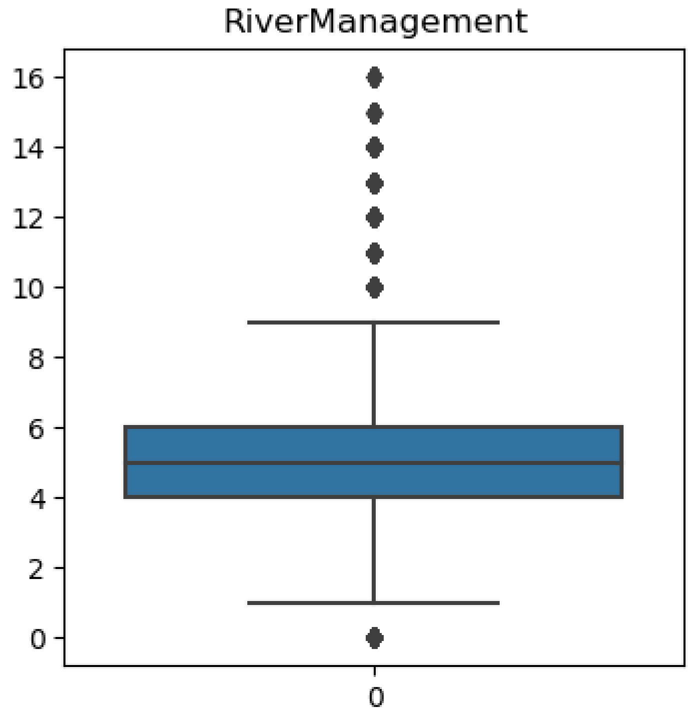
```
Axes(0.125,0.11;0.775x0.77)
```



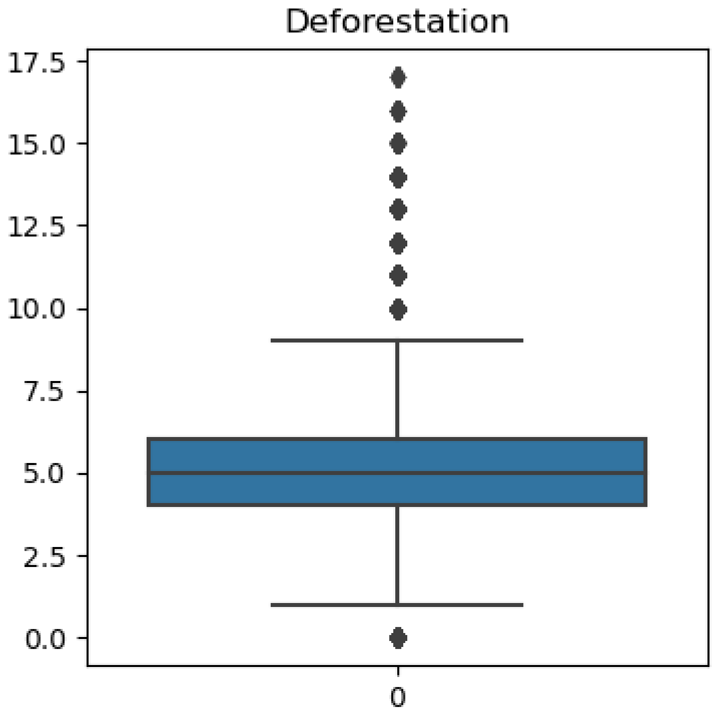

```
Axes(0.125,0.11;0.775x0.77)
```



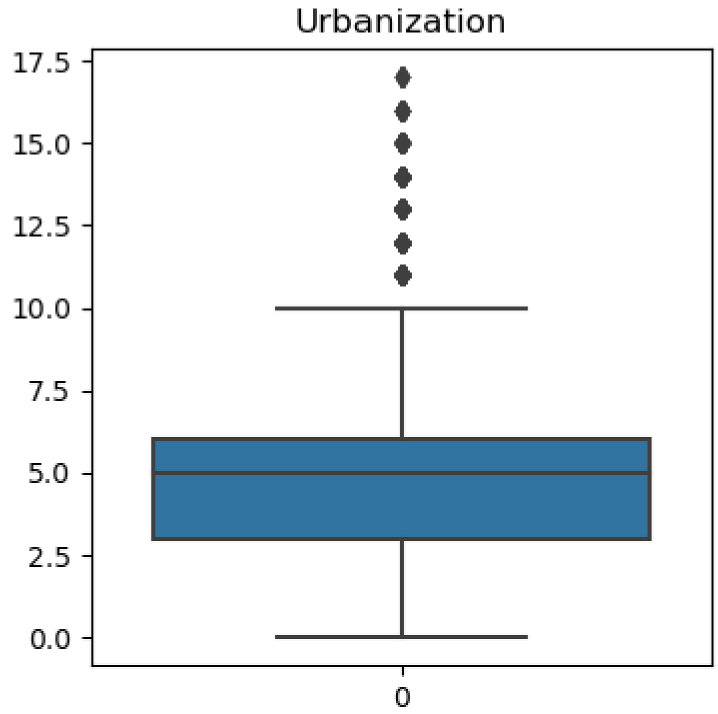

```
Axes(0.125,0.11;0.775x0.77)
```



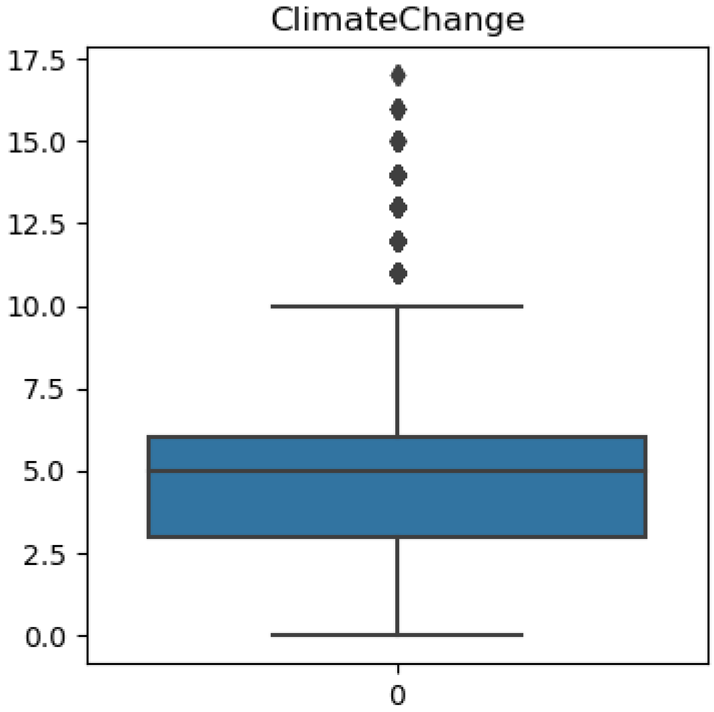

```
Axes(0.125,0.11;0.775x0.77)
```



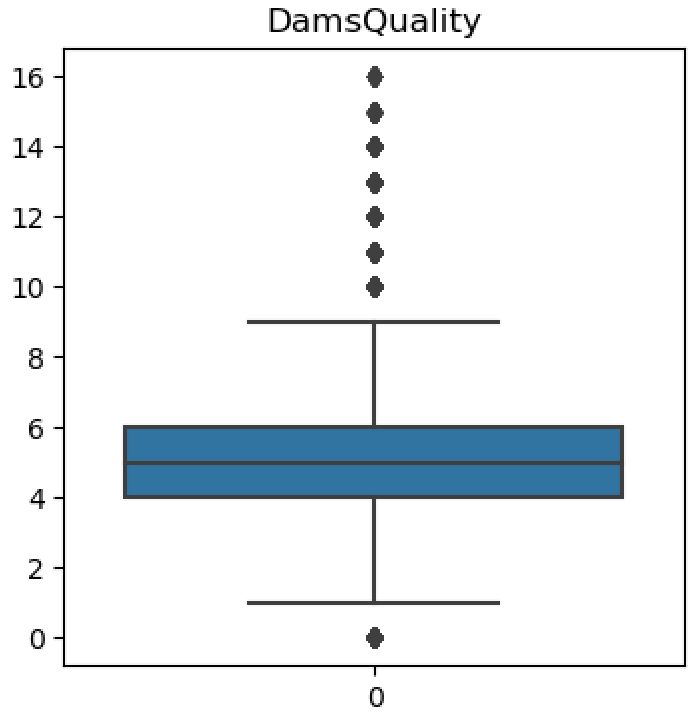

```
Axes(0.125,0.11;0.775x0.77)
```



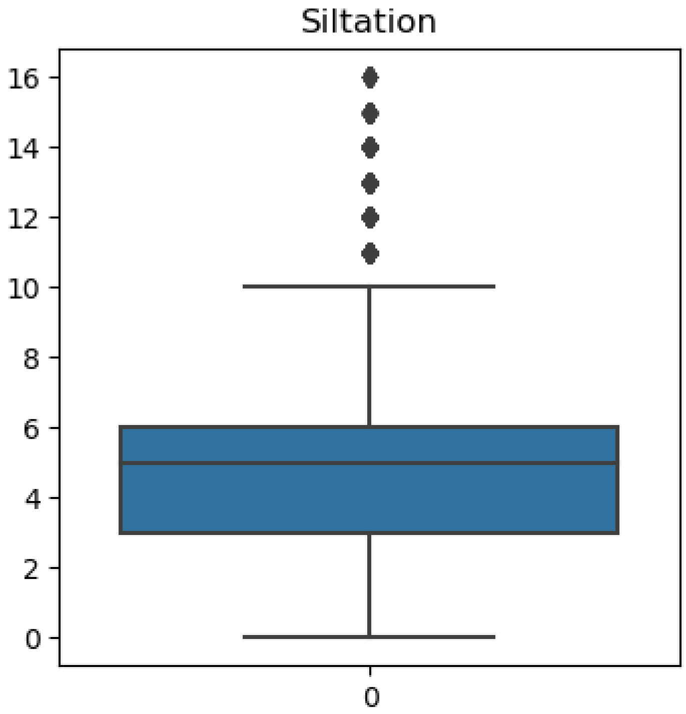

```
Axes(0.125,0.11;0.775x0.77)
```



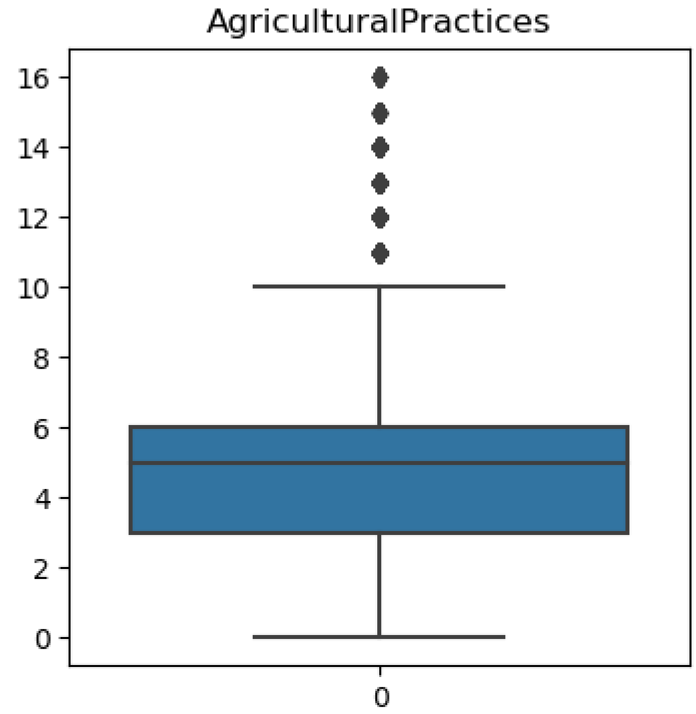

```
Axes(0.125,0.11;0.775x0.77)
```



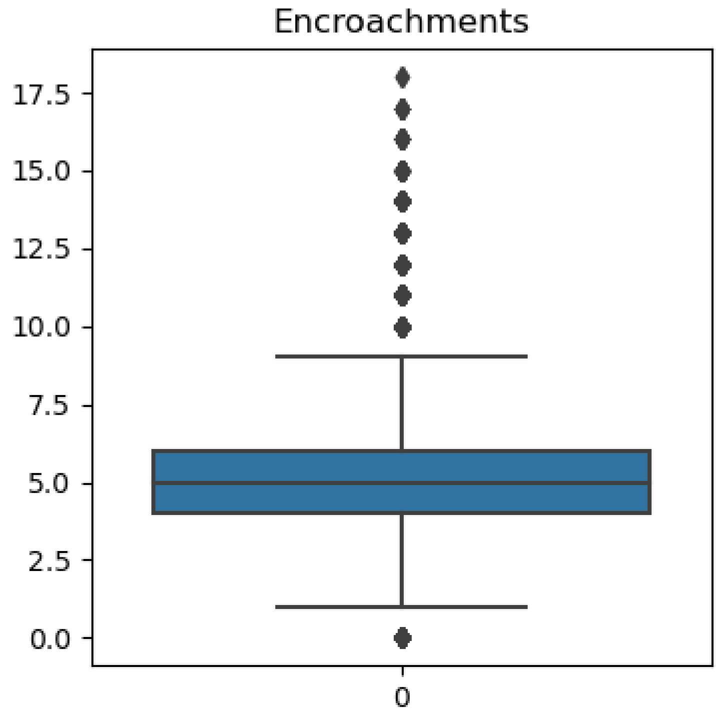

```
Axes(0.125,0.11;0.775x0.77)
```



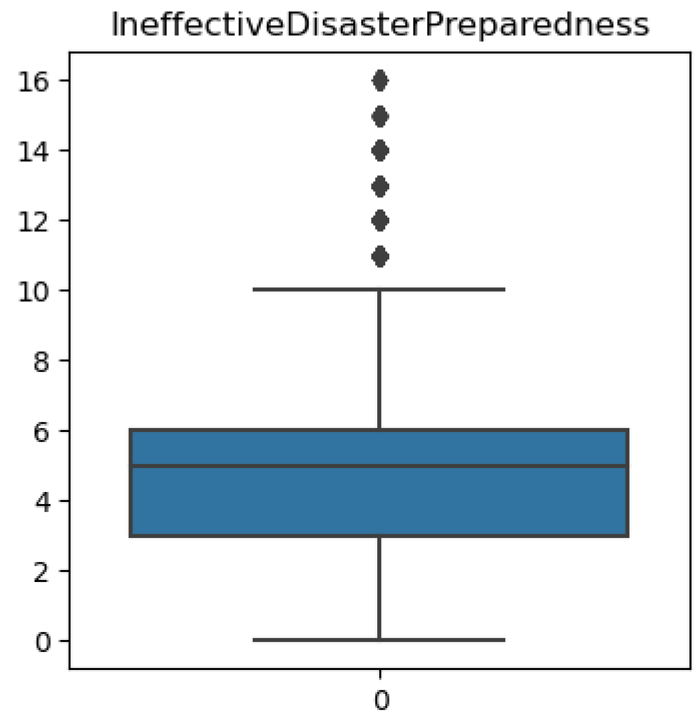

```
Axes(0.125,0.11;0.775x0.77)
```



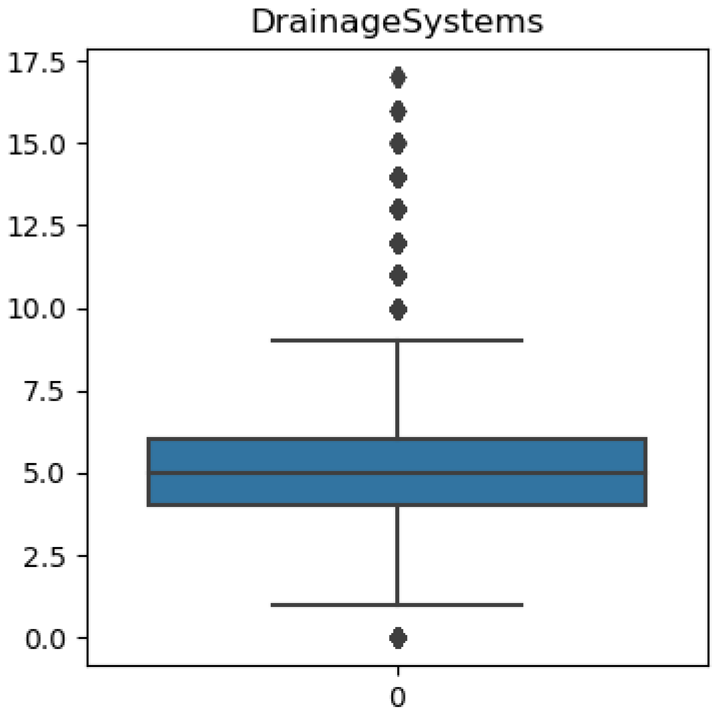

```
Axes(0.125,0.11;0.775x0.77)
```



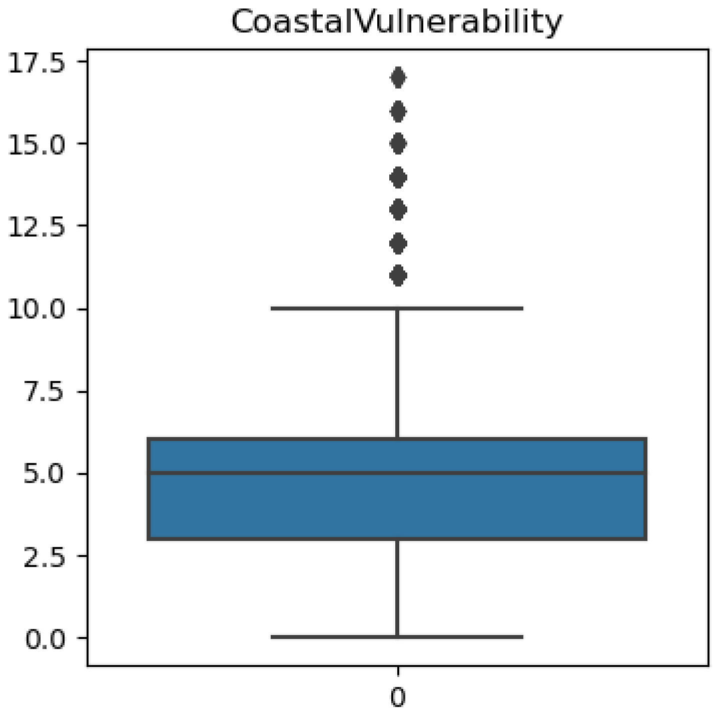

```
Axes(0.125,0.11;0.775x0.77)
```



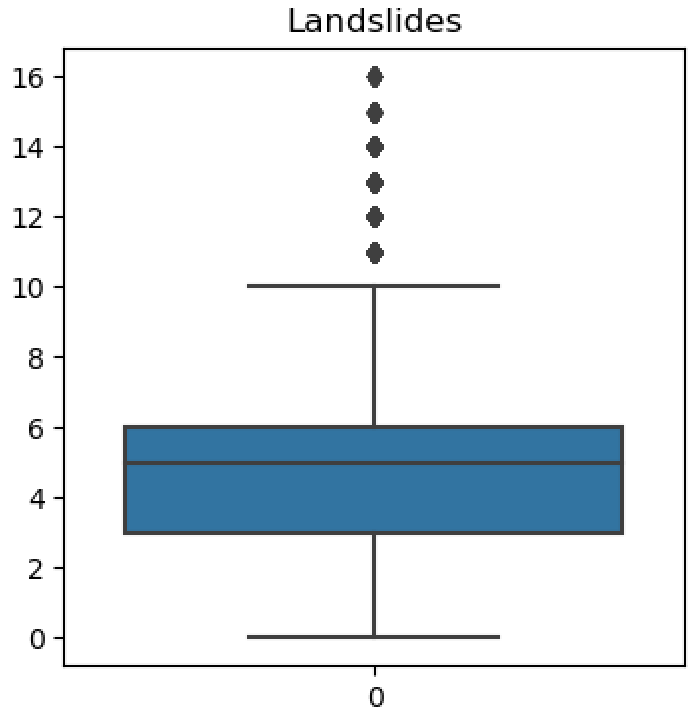

```
Axes(0.125,0.11;0.775x0.77)
```



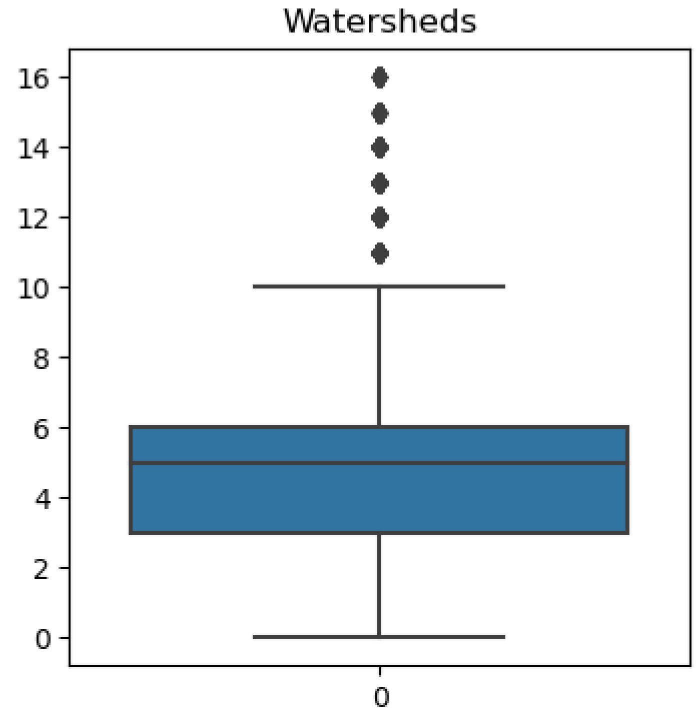

```
Axes(0.125,0.11;0.775x0.77)
```



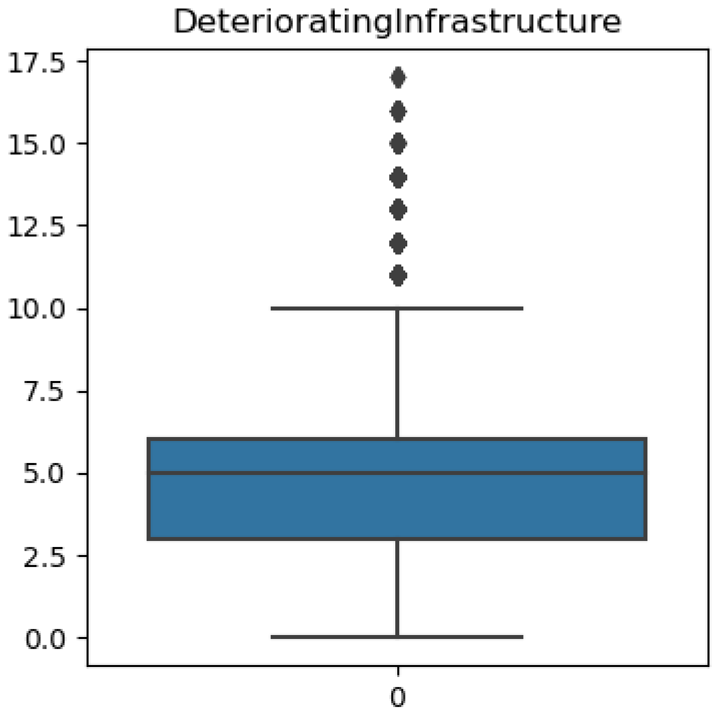

```
Axes(0.125,0.11;0.775x0.77)
```



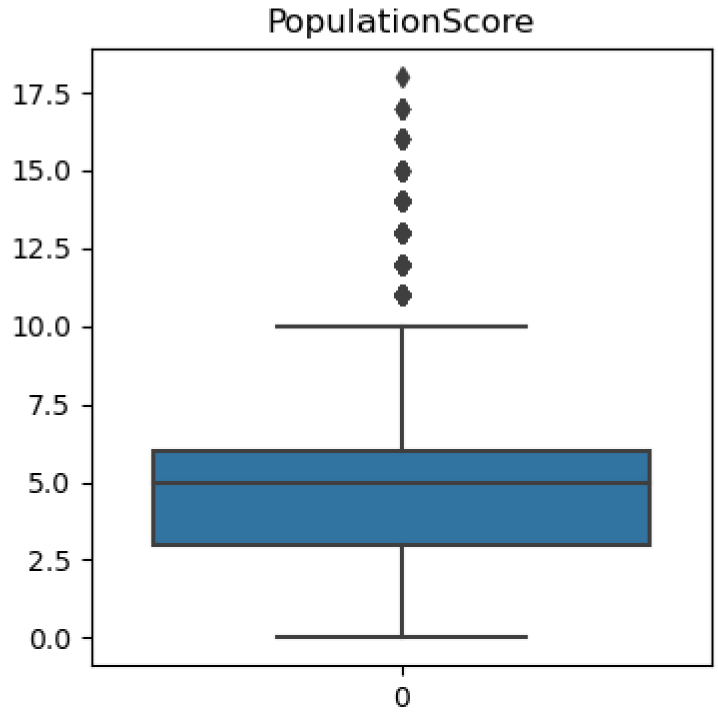

```
Axes(0.125,0.11;0.775x0.77)
```



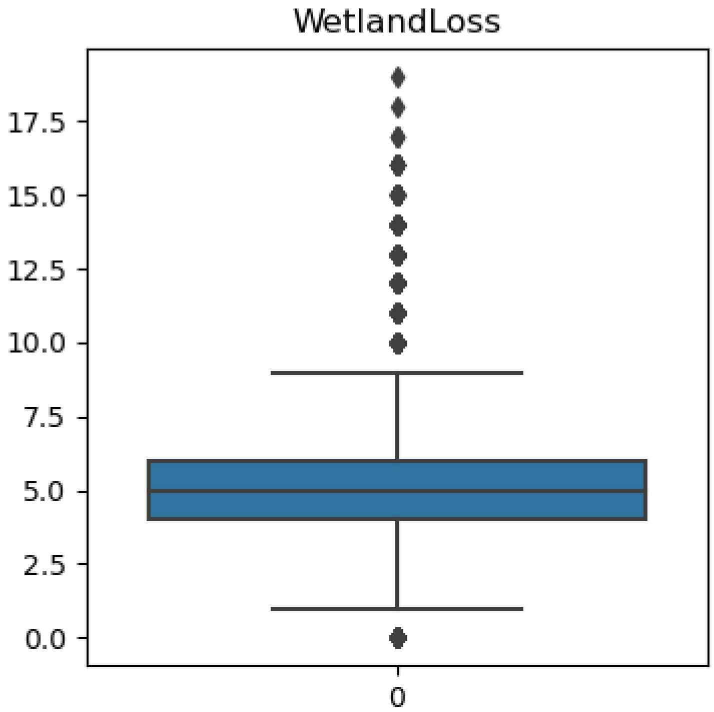

```
Axes(0.125,0.11;0.775x0.77)
```



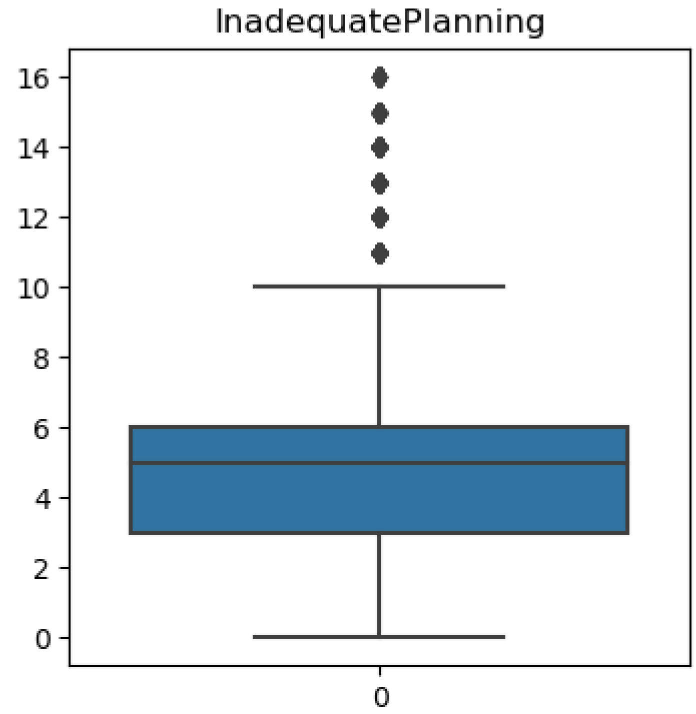

```
Axes(0.125,0.11;0.775x0.77)
```



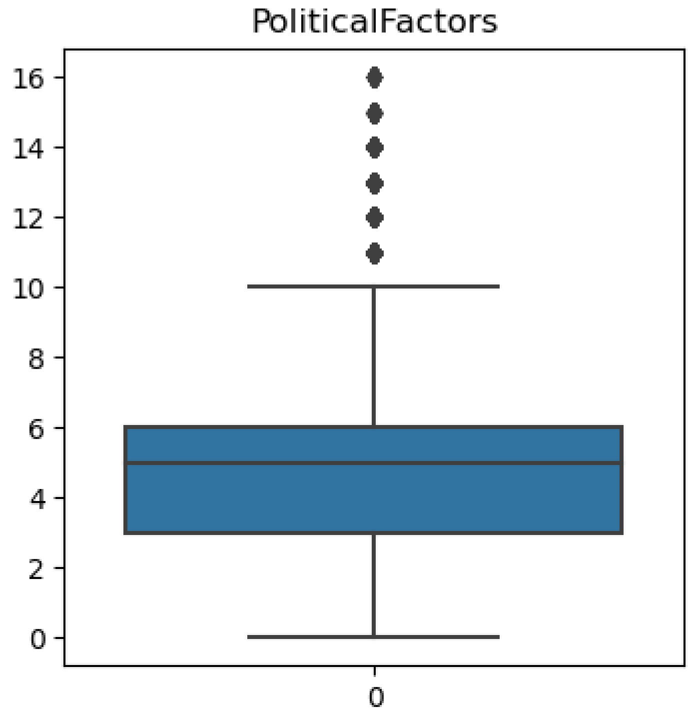

```
Axes(0.125,0.11;0.775x0.77)
```



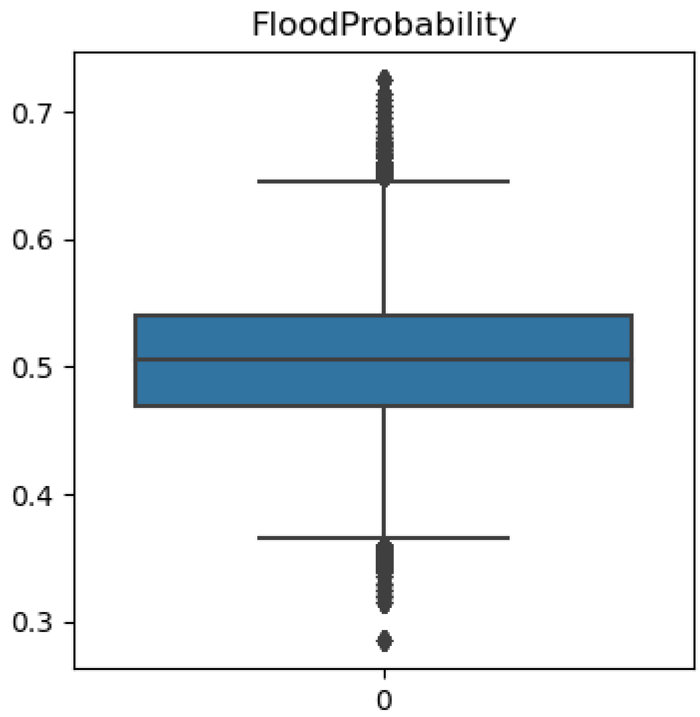

```
Axes(0.125,0.11;0.775x0.77)
```




```
Axes(0.125,0.11;0.775x0.77)
```




```
Axes(0.125,0.11;0.775x0.77)
```



3.2 Multivariate Analysis

3.2.2 Correlation Matrix

In [15]:
```
df_train.corr()
```

Out[15]:

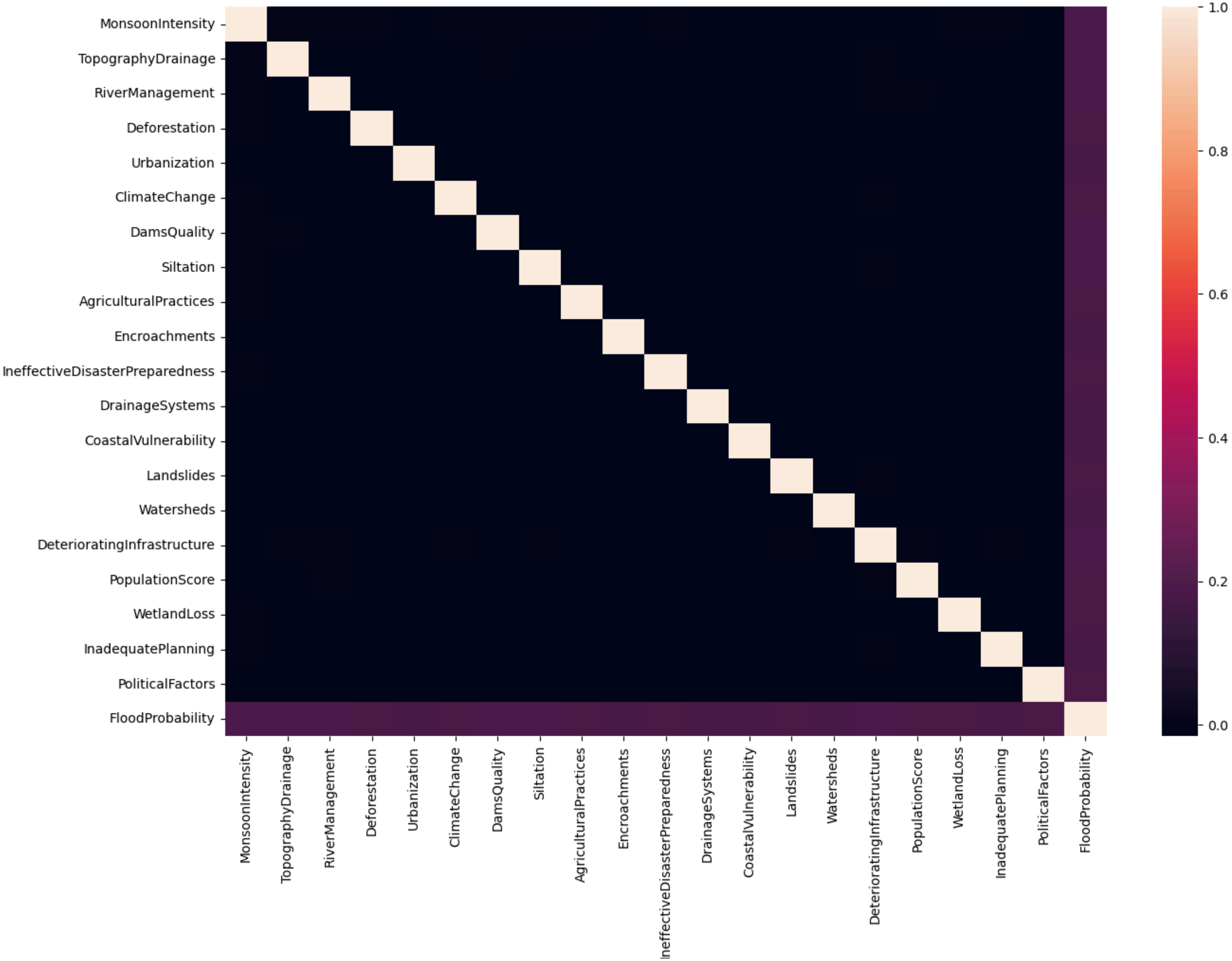
	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	S
MonsoonIntensity	1.000000	-0.007362	-0.008070	-0.007251	-0.009309	-0.008031	-0.007787	-0
TopographyDrainage	-0.007362	1.000000	-0.009924	-0.008548	-0.010532	-0.009619	-0.007607	-0
RiverManagement	-0.008070	-0.009924	1.000000	-0.008574	-0.012292	-0.009237	-0.008711	-0
Deforestation	-0.007251	-0.008548	-0.008574	1.000000	-0.012248	-0.008266	-0.009356	-0
Urbanization	-0.009309	-0.010532	-0.012292	-0.012248	1.000000	-0.011199	-0.011128	-0
ClimateChange	-0.008031	-0.009619	-0.009237	-0.008266	-0.011199	1.000000	-0.008427	-0
DamsQuality	-0.007787	-0.007607	-0.008711	-0.009356	-0.011128	-0.008427	1.000000	-0
Siltation	-0.007836	-0.009824	-0.010058	-0.011536	-0.010153	-0.009457	-0.009401	1
AgriculturalPractices	-0.008232	-0.009496	-0.010783	-0.010039	-0.010559	-0.011517	-0.009033	-0
Encroachments	-0.010309	-0.012887	-0.011615	-0.013175	-0.010784	-0.012533	-0.010890	-0
IneffectiveDisasterPreparedness	-0.008032	-0.010746	-0.010675	-0.009512	-0.012685	-0.011346	-0.009515	-0
DrainageSystems	-0.009716	-0.010056	-0.011277	-0.010490	-0.012572	-0.009650	-0.010439	-0
CoastalVulnerability	-0.010659	-0.012526	-0.011680	-0.012388	-0.014497	-0.013005	-0.012096	-0
Landslides	-0.009121	-0.010240	-0.008994	-0.009257	-0.010582	-0.009352	-0.009924	-0
Watersheds	-0.008900	-0.011067	-0.011412	-0.010671	-0.012107	-0.009882	-0.009085	-0
DeterioratingInfrastructure	-0.008486	-0.006628	-0.005827	-0.008862	-0.010656	-0.006324	-0.009831	-0
PopulationScore	-0.008679	-0.010815	-0.006727	-0.011777	-0.011485	-0.010332	-0.009599	-0
WetlandLoss	-0.006811	-0.010267	-0.010069	-0.011004	-0.011023	-0.009376	-0.009372	-0
InadequatePlanning	-0.008155	-0.011617	-0.009673	-0.010424	-0.011584	-0.010772	-0.011374	-0
PoliticalFactors	-0.008474	-0.012350	-0.011550	-0.009661	-0.013005	-0.011379	-0.013081	-0
FloodProbability	0.189098	0.187635	0.187131	0.184001	0.180861	0.184761	0.187996	0

21 rows × 21 columns

Remark: As correlation coefficients are tends to `0`, no `dependent feature` exists

3.2.3 Correlation Heatmap

In [16]:
```
plt.figure(figsize =(15,10))
sns.heatmap(df_train.corr())
plt.show()
```


MonsoonIntensity
TopographyDrainage
RiverManagement
Deforestation
Urbanization
ClimateChange
DamsQuality
Siltation
AgriculturalPractices
Encroachments
IneffectiveDisasterPreparedness
DrainageSystems
CoastalVulnerability
Landslides
Watersheds
DeterioratingInfrastructure
PopulationScore
WetlandLoss
InadequatePlanning
PoliticalFactors
FloodProbability
1.0
0.8
0.6
0.4
0.2
0.0

4. Feature Engineering

4.1 Feature Transformation

4.1.2 Scaling

4.1.2.1 Standardization

In [17]:
```
x_train = df_train.iloc[:,:20]
y_train = df_train.iloc[:,-1]
x_test = df_test.iloc[:,1:]
```

In [18]:
```
x_train = (x_train - x_train.mean())/x_train.std()
x_test = (x_test - x_test.mean())/x_test.std()
```

In [19]:
```
x_train.head(2)
```

Out[19]:

	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	AgriculturalPractices
0	0.038198	1.467768	0.021561	1.490362	0.507578	-0.453941	-0.458881	-0.933107	-0.939124
1	0.524488	0.990186	-0.461021	-0.459251	1.467551	1.489938	-0.938943	0.034951	-0.455692

4.2 Feature Extreaction

4.2.1 Principal Component Analysis

In [20]:
```
from sklearn.decomposition import PCA
```

In [21]:
```
x_trn = x_train

pca = PCA()
x_trn = pca.fit_transform(x_train)
```

```
explained_variance = pca.explained_variance_ratio_
print(explained_variance)
```

```
[0.05098534 0.05087619 0.0508146  0.05078321 0.0506943  0.05066361
 0.05063026 0.05061952 0.05058528 0.05051905 0.05045643 0.05041462
 0.05038968 0.05034189 0.0503254  0.05029073 0.05025888 0.05023953
 0.05011    0.04000149]
```

Remark : As it can be observed that for `95%` proportion of variance explained, the number of principal components are `20`. So, it is not suggested to go for `PCA`, as it will not reduce a significant dimension.

4.3 Feature Selection

4.3.1 Filter Method (Pearson Correlation Coefficient)

In [22]:
```
for i in x_train.columns:
    if np.corrcoef(x_train[i],y_train)[1][0] > 0.56:
        print(i,"is poorly related to `Target Variable","Corr_coeff :",np.corrcoef(x_train[i],y_train)[1][0])
else:
    print("All are considered to be significant for now.")
```

```
All are considered to be significant for now.
```

Remark: As, `no` independent variable is `highly` related to dependent variable, all are taken into consideration.

5. Model Building

5.1 Splitting `Training data` into `Model Training` and `Hold data` (For Model Training)

In [23]:
```
from sklearn.model_selection import train_test_split
x_trn,x_tst,y_trn,y_tst=train_test_split(x_train,y_train,test_size=0.25)
```

5.1.1 Shape of Training and Test Data for Models

In [24]:
```
x_trn.shape,x_tst.shape,y_trn.shape,y_tst.shape
```

Out[24]:
```
((838467, 20), (279490, 20), (838467,), (279490,))
```

5.2 Model Building

5.2.1 Importing Libraries

In [29]:
```
from sklearn.ensemble import GradientBoostingRegressor
import xgboost as xgb
import lightgbm as lgb
import catboost as cb
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingRegressor
```

5.2.2 Baseline Models

In [30]:
```
base_model_gb = GradientBoostingRegressor(random_state=42,max_depth=10)
base_model_xgb = xgb.XGBRegressor(random_state=42)
base_model_lgb = lgb.LGBMRegressor(random_state=42)
base_model_cb = cb.CatBoostRegressor(random_state=42, verbose=False)
base_model_hgb = HistGradientBoostingRegressor(random_state=42,)
from sklearn.model_selection import GridSearchCV
```

5.2.2.1 GradientBoostingRegressor

5.2.3 Model Accuracy of each Base models

In [31]:
```
from sklearn.metrics import mean_squared_error,r2_score
```

5.2.3.1 Model Accuracy of GradientBoostingRegressor

In [32]:
```
base_model_gb.fit(x_trn, y_trn)
y_prd_1 = base_model_gb.predict(x_tst)
mse_trn_1 = mean_squared_error(y_tst, y_prd_1)
r2_score_trn_1 = r2_score(y_tst, y_prd_1)
print("Gradient Boosting MSE:", mse_trn_1)
print("Gradient Boosting r2_score:", r2_score_trn_1)
```

```
Gradient Boosting MSE: 0.000510660798632034
Gradient Boosting r2_score: 0.8042536260425572
```

5.2.3.2 Model Accuracy of XGBoostingRegressor

In [33]:
```
base_model_xgb.fit(x_trn, y_trn)
y_prd_2 = base_model_xgb.predict(x_tst)
mse_trn_2 = mean_squared_error(y_tst, y_prd_2)
r2_score_trn_2 = r2_score(y_tst, y_prd_2)
print("XG Boosting MSE:", mse_trn_2)
print("XG r2_score:", r2_score_trn_2)
```

```
XG Boosting MSE: 0.0004971473252938768
XG r2_score: 0.8094336073777232
```

5.2.3.3 Model Accuracy of LGBMRegressor

In [34]:
```
base_model_lgb.fit(x_trn, y_trn)
y_prd_3 = base_model_lgb.predict(x_tst)
mse_trn_3 = mean_squared_error(y_tst, y_prd_3)
r2_score_trn_3 = r2_score(y_tst, y_prd_3)
print("lg Boosting MSE:", mse_trn_3)
print("lg r2_score:", r2_score_trn_3)
```

```
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.010661 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 369
[LightGBM] [Info] Number of data points in the train set: 838467, number of used features: 20
[LightGBM] [Info] Start training from score 0.504498
lg Boosting MSE: 0.0006103419392396314
lg r2_score: 0.7660438753075315
```

5.2.3.4 Model Accuracy of CatBoostRegressor

In [35]:
```
base_model_cb.fit(x_trn, y_trn)
y_prd_4 = base_model_cb.predict(x_tst)
mse_trn_4 = mean_squared_error(y_tst, y_prd_4)
r2_score_trn_4 = r2_score(y_tst, y_prd_4)
print("cat Boosting MSE:", mse_trn_4)
print("cat r2_score:", r2_score_trn_4)
```

```
cat Boosting MSE: 0.00040076790972 07242
cat r2_score: 0.8463777416702328
```

5.2.3.5 Model Accuracy of HistGradientBoostingRegressor

In [36]:
```
base_model_hgb.fit(x_trn, y_trn)
y_prd_5 = base_model_hgb.predict(x_tst)
mse_trn_5 = mean_squared_error(y_tst, y_prd_5)
r2_score_trn_5 = r2_score(y_tst, y_prd_5)
print("Hist Gradient Boosting MSE:", mse_trn_5)
print("Hist Gradient r2_score:", r2_score_trn_5)
```

```
Hist Gradient Boosting MSE: 0.00061027484237711
Hist Gradient r2_score: 0.7660695948606628
```

5.2.4 Voting Regressor

In [37]:
```
from sklearn.ensemble import VotingRegressor
```

In [38]:
```
voting_regressor = VotingRegressor(estimators=[
    ('gb', base_model_gb),
    ('xgb', base_model_xgb),
    ('lgb', base_model_lgb),
    ('cb', base_model_cb),
    ('hgb', base_model_hgb)
],weights=[r2_score_trn_1,r2_score_trn_2,r2_score_trn_3,r2_score_trn_4,r2_score_trn_5])
```

5.2.5 Final Model

In [39]:
```
voting_regressor.fit(x_trn, y_trn)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.031670 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 369
[LightGBM] [Info] Number of data points in the train set: 838467, number of used features: 20
[LightGBM] [Info] Start training from score 0.504498
```

Out[39]:

```
VotingRegressor
gb                          xgb            lgb             cb                hgb
GradientBoostingRegressor   XGBRegressor   LGBMRegressor   CatBoostRegressor HistGradientBoostingRegressor
```

6.Model Accuracy

6.1 Model Accuracy on `Hold data` (Model Training Data)

In [40]:

```
y_prd = voting_regressor.predict(x_tst)
mse_trn = mean_squared_error(y_tst, y_prd)
r2_score_trn = r2_score(y_tst, y_prd)
print("Final model MSE:", mse_trn)
print("Final model r2_score:", r2_score_trn)
```

```
Final model MSE: 0.00046095654976258175
Final model r2_score: 0.8233062467107912
```

7. Model Prediction

7.1 Model Prediction on `Test data` (Model Testing Data)

In [41]:

```
y_test = voting_regressor.predict(x_test)
```

In [46]:

```
y_test
```

Out[46]:

```
array([0.55586725, 0.44691242, 0.45401758, ..., 0.59006637, 0.53860149,
       0.49536748])
```

7.2 Submission

7.2.1 Addition of `Id` in final submission

In [42]:

```
df_test.head()
```

Out[42]:

	id	MonsoonIntensity	TopographyDrainage	RiverManagement	Deforestation	Urbanization	ClimateChange	DamsQuality	Siltation	Agricultural
0	1117957	4	6	3	5	6	7	8	7	
1	1117958	4	4	2	9	5	5	4	7	
2	1117959	1	3	6	5	7	2	4	6	
3	1117960	2	4	4	6	4	5	4	3	
4	1117961	6	3	2	4	6	4	5	5	

5 rows × 21 columns

In [43]:

```
submission_df = pd.DataFrame({
    'id': df_test['id'],
    'FoodPrediction': y_test
})
```

In [44]:

```
submission_df.head()
```

Out[44]:

	id	FoodPrediction
0	1117957	0.555867
1	1117958	0.446912
2	1117959	0.454018
3	1117960	0.456258
4	1117961	0.457879

In [45]:

```
submission_df.to_csv('submission1.csv', index=False)
print("Submission file created successfully.")
```

```
Submission file created successfully.
```