County level population by race ethicity 2010-2019

In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import PowerTransformer
from sklearn.feature_selection import SelectKBest,f_classif,f_regression
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.svm import SVR
```

Data Collection

In [4]:
```
# importing library
import numpy as np
import pandas as pd
```

In [5]:
```
# Reading downloded Dataset from loacl directory
data = pd.read_csv('US_county_census_est_race_eth_2010_2019.csv')
# Converging Data to panda DataFrame
df = pd.DataFrame(data)
```

In [6]:
```
## For result and title printing, create a custom definition.
def print_title(title):
    print(f'\n{'-'*60}\n\033[1m{title}\033[0m')
def print_section(title):
    print(f'{'-'*60}\n{title}\n{'-'*60}')
```

In [7]:
```
df.head(3)
```

Out[7]:

	FIPS	STFIPS	COFIPS	state_abbrev	state	county	year	pop	white_pop	black_pop	asian_pop	indian
0	1001	1	1	AL	Alabama	Autauga	2010	54571	43297	9689	484	
1	1001	1	1	AL	Alabama	Autauga	2011	55227	43699	9883	514	
2	1001	1	1	AL	Alabama	Autauga	2012	54954	43315	9949	552	

In [8]:
```
df.tail(3)
```

Out[8]:

	FIPS	STFIPS	COFIPS	state_abbrev	state	county	year	pop	white_pop	black_pop	asian_pop	ir
31407	56045	56	45	WY	Wyoming	Weston	2017	6968	6558	44	97	
31408	56045	56	45	WY	Wyoming	Weston	2018	6924	6474	47	109	
31409	56045	56	45	WY	Wyoming	Weston	2019	6927	6454	48	117	

In [9]:
```
df1 = pd.DataFrame(df)
print_section(f'Since column "state_abbrev" and "state" columns are same, \ndorping "sta
df1 = df1.drop("state_abbrev", axis=1)
```

```
print_section("DataFrame after dropping the column")
df1.head(3)
```

```
---------------------------------------------------------------
Since column "state_abbrev" and "state" columns are same,
dorping "state_abbrev" columns and creating df1 the from datafrme
---------------------------------------------------------------
---------------------------------------------------------------
DataFrame after dropping the column
---------------------------------------------------------------
```

Out[9]:

	FIPS	STFIPS	COFIPS	state	county	year	pop	white_pop	black_pop	asian_pop	indian_pop	pacific_
0	1001	1	1	Alabama	Autauga	2010	54571	43297	9689	484	258	
1	1001	1	1	Alabama	Autauga	2011	55227	43699	9883	514	261	
2	1001	1	1	Alabama	Autauga	2012	54954	43315	9949	552	275	

In [10]:

```
print_title("DataFrame Information")
print_section(df1.info())
```

```
---------------------------------------------------------------
DataFrame Information
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31410 entries, 0 to 31409
Data columns (total 15 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   FIPS          31410 non-null  int64 
 1   STFIPS        31410 non-null  int64 
 2   COFIPS        31410 non-null  int64 
 3   state         31410 non-null  object
 4   county        31410 non-null  object
 5   year          31410 non-null  int64 
 6   pop           31410 non-null  int64 
 7   white_pop     31410 non-null  int64 
 8   black_pop     31410 non-null  int64 
 9   asian_pop     31410 non-null  int64 
 10  indian_pop    31410 non-null  int64 
 11  pacific_pop   31410 non-null  int64 
 12  two_pop       31410 non-null  int64 
 13  not_hisp_pop  31410 non-null  int64 
 14  hisp_pop      31410 non-null  int64 
dtypes: int64(13), object(2)
memory usage: 3.6+ MB
---------------------------------------------------------------
None
---------------------------------------------------------------
```

In [11]:

```
print_title("DataFrame Description")
print_section(df1.describe())
```

```
---------------------------------------------------------------
DataFrame Description
---------------------------------------------------------------
               FIPS        STFIPS        COFIPS          year           pop  \
count  31410.000000  31410.000000  31410.000000  31410.000000  3.141000e+04   
mean   30389.820121     30.286215    103.605540   2014.500000  1.014097e+05   
std    15158.803727     15.140671    107.690218      2.872327  3.251245e+05   
min     1001.000000      1.000000      1.000000   2010.000000  8.200000e+01   
25%    18179.000000     18.000000     35.000000   2012.000000  1.098500e+04   
50%    29177.000000     29.000000     79.000000   2014.500000  2.573350e+04   
75%    45081.000000     45.000000    133.000000   2017.000000  6.741675e+04   
max    56045.000000     56.000000    840.000000   2019.000000  1.010571e+07   

          white_pop     black_pop    asian_pop    indian_pop   pacific_pop  \
```

```
count   3.141000e+04  3.141000e+04  3.141000e+04   31410.000000  31410.000000
mean    7.844218e+04  1.335439e+04  5.543686e+03    1264.135371    236.176313
std     2.333952e+05  5.778493e+04  4.089464e+04    5203.210936   2150.884073
min     2.400000e+01  0.000000e+00  0.000000e+00       0.000000      0.000000
25%     9.105000e+03  1.170000e+02  4.700000e+01      64.000000      4.000000
50%     2.217200e+04  8.400000e+02  1.560000e+02     179.000000     14.000000
75%     5.846450e+04  5.697750e+03  7.750000e+02     618.750000     60.000000
max     7.181207e+06  1.311698e+06  1.545445e+06  146005.000000  95285.000000

             two_pop  not_hisp_pop      hisp_pop
count   31410.000000  3.141000e+04  3.141000e+04
mean     2569.121999  8.370397e+04  1.770571e+04
std     10265.356718  2.222491e+05  1.228180e+05
min         0.000000  6.400000e+01  0.000000e+00
25%       156.000000  9.944500e+03  3.240000e+02
50%       392.000000  2.386300e+04  1.015500e+03
75%      1348.750000  6.290925e+04  4.764500e+03
max    315568.000000  5.211947e+06  4.899383e+06
------------------------------------------------------------
```

In [12]:

```python
print_title("Null values in DataFrame")
print_section(df1.isnull().sum())
```

```
------------------------------------------------------------
Null values in DataFrame
------------------------------------------------------------
FIPS            0
STFIPS          0
COFIPS          0
state           0
county          0
year            0
pop             0
white_pop       0
black_pop       0
asian_pop       0
indian_pop      0
pacific_pop     0
two_pop         0
not_hisp_pop    0
hisp_pop        0
dtype: int64
------------------------------------------------------------
```

In [13]:

```python
print_title("Data Types of Dataframe Varible columns")
print_section(df1.dtypes)
```

```
------------------------------------------------------------
Data Types of Dataframe Varible columns
------------------------------------------------------------
FIPS             int64
STFIPS           int64
COFIPS           int64
state           object
county          object
year             int64
pop              int64
white_pop        int64
black_pop        int64
asian_pop        int64
indian_pop       int64
pacific_pop      int64
two_pop          int64
not_hisp_pop     int64
hisp_pop         int64
```

```
dtype: object
------------------------------------------------------------
```

In [14]:
```
print_title("Duplicated values")
print_section(df1.duplicated().sum())
```

```
------------------------------------------------------------
Duplicated values
------------------------------------------------------------
0
------------------------------------------------------------
```

Encoding

Encoding of object-type columns

In [16]:
```
# importing category encoder from library
from category_encoders import TargetEncoder
```

In [17]:
```
df2 = pd.DataFrame(df1)
## finding Object type columns and count the number of unique values
col_object_type=df2.select_dtypes(include=['object']).columns.tolist()
print_title('Object type columns in DataFrame')
print_section(pd.DataFrame({'Colums':col_object_type}))
print_title('Counted unique values in object type columns')
for column_name in col_object_type:
  print(f'{column_name} has {df2[column_name].nunique()} unique values')
```

```
------------------------------------------------------------
Object type columns in DataFrame
------------------------------------------------------------
   Colums
0   state
1  county
------------------------------------------------------------

------------------------------------------------------------
Counted unique values in object type columns
state has 50 unique values
county has 1876 unique values
```

In [18]:
```
print_title('Aggrigation of Cont and Mean of object type column to target column')
for column_name in col_object_type:
    print_section(df2['pop'].groupby(df2[column_name]).agg(['count','mean']))
```

```
------------------------------------------------------------
Aggrigation of Cont and Mean of object type column to target column
------------------------------------------------------------
                count           mean
state
Alabama           670   72309.697015
Alaska            290   25246.520690
Arizona           150  453562.080000
Arkansas          750   39643.185333
California        580  665717.851724
Colorado          640   84396.614062
Connecticut        80  447770.737500
Delaware           30  312093.200000
Florida           670  300077.741791
Georgia          1590   63805.340881
Hawaii             50  281353.680000
Idaho             440   37677.020455
Illinois         1020  125699.390196
Indiana           920   71770.554348
Iowa              990   31402.825253
```

```
Kansas           1050   27575.745714
Kentucky         1200   36799.828333
Louisiana         640   72344.243750
Maine             160   83252.225000
Maryland          240  247811.120833
Massachusetts     140  482520.650000
Michigan          830  119677.637349
Minnesota         870   62882.937931
Mississippi       820   36379.403659
Missouri         1150   52734.846957
Montana           560   18355.733929
Nebraska          930   20255.675269
Nevada            170  168310.464706
New Hampshire     100  133614.870000
New Jersey        210  421804.600000
New Mexico        330   63245.618182
New York          620  315463.466129
North Carolina   1000  100043.140000
North Dakota      530   13777.383019
Ohio              880  131916.434091
Oklahoma          770   50330.257143
Oregon            360  111491.219444
Pennsylvania      670  190653.114925
Rhode Island       50  211160.200000
South Carolina    460  105883.593478
South Dakota      660   12902.998485
Tennessee         950   69243.832632
Texas            2540  106938.801575
Utah              290  102587.641379
Vermont           140   44656.200000
Virginia         1330   62493.806767
Washington        390  183301.069231
West Virginia     550   33374.700000
Wisconsin         720   79936.259722
Wyoming           230   25117.400000
------------------------------------------------------------
------------------------------------------------------------
                           count       mean
county
Abbeville                     10   24834.500
Acadia Parish                 10   62288.400
Accomack                      10   32896.400
Ada                           10  432815.800
Adair                         40   18568.025
...                          ...         ...
Yukon-Koyukuk Census Area     10    5474.700
Yuma                          20  107774.500
Zapata                        10   14280.300
Zavala                        10   12020.100
Ziebach                       10    2818.400

[1876 rows x 2 columns]
------------------------------------------------------------
```

since high cardinality in Features TargetEncoding method used

In [20]:
```
print_section(f'Since high cardinality in state and county coluns Target\nEncoding is mo
encoder=TargetEncoder()
```

```
------------------------------------------------------------
Since high cardinality in state and county coluns Target
Encoding is moste prefered
------------------------------------------------------------
```

In [21]:
```
df3 = pd.DataFrame(df2)
```

```
df3.head(2)
```

Out[21]:

	FIPS	STFIPS	COFIPS	state	county	year	pop	white_pop	black_pop	asian_pop	indian_pop	pacific_
0	1001	1	1	Alabama	Autauga	2010	54571	43297	9689	484	258	
1	1001	1	1	Alabama	Autauga	2011	55227	43699	9883	514	261	

In [22]:

```
print_section(f'Since high cardinality in state and county coluns Target\nEncoding is mo
encoder.fit(df2['state'],df2['pop'])
df3['state'] = encoder.transform(df2['state'],df2['pop'])
encoder.fit(df2['county'],df2['pop'])
df3['county'] = encoder.transform(df2['county'],df2['pop'])
df3.head(3)
```

```
------------------------------------------------------------
Since high cardinality in state and county coluns Target
Encoding is moste prefered
------------------------------------------------------------
```

Out[22]:

	FIPS	STFIPS	COFIPS	state	county	year	pop	white_pop	black_pop	asian_pop	indian_pop
0	1001	1	1	72309.697015	88962.379901	2010	54571	43297	9689	484	258
1	1001	1	1	72309.697015	88962.379901	2011	55227	43699	9883	514	261
2	1001	1	1	72309.697015	88962.379901	2012	54954	43315	9949	552	275

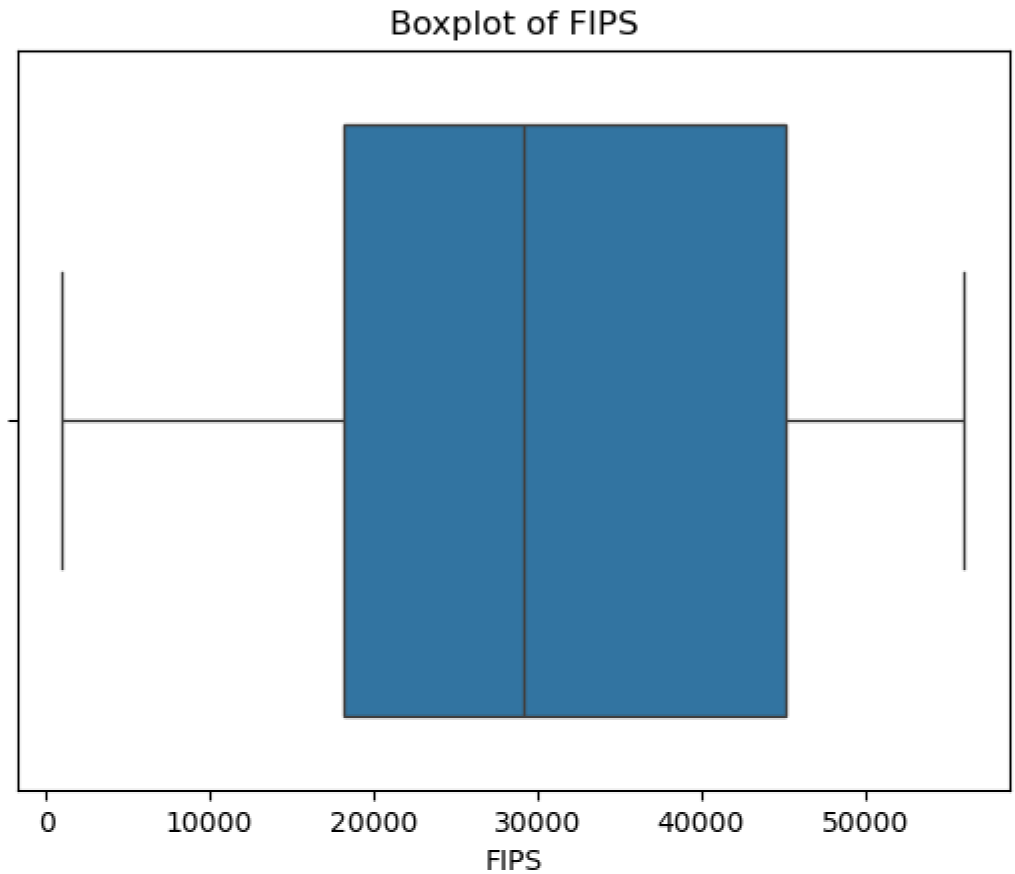
Outlayer removeal

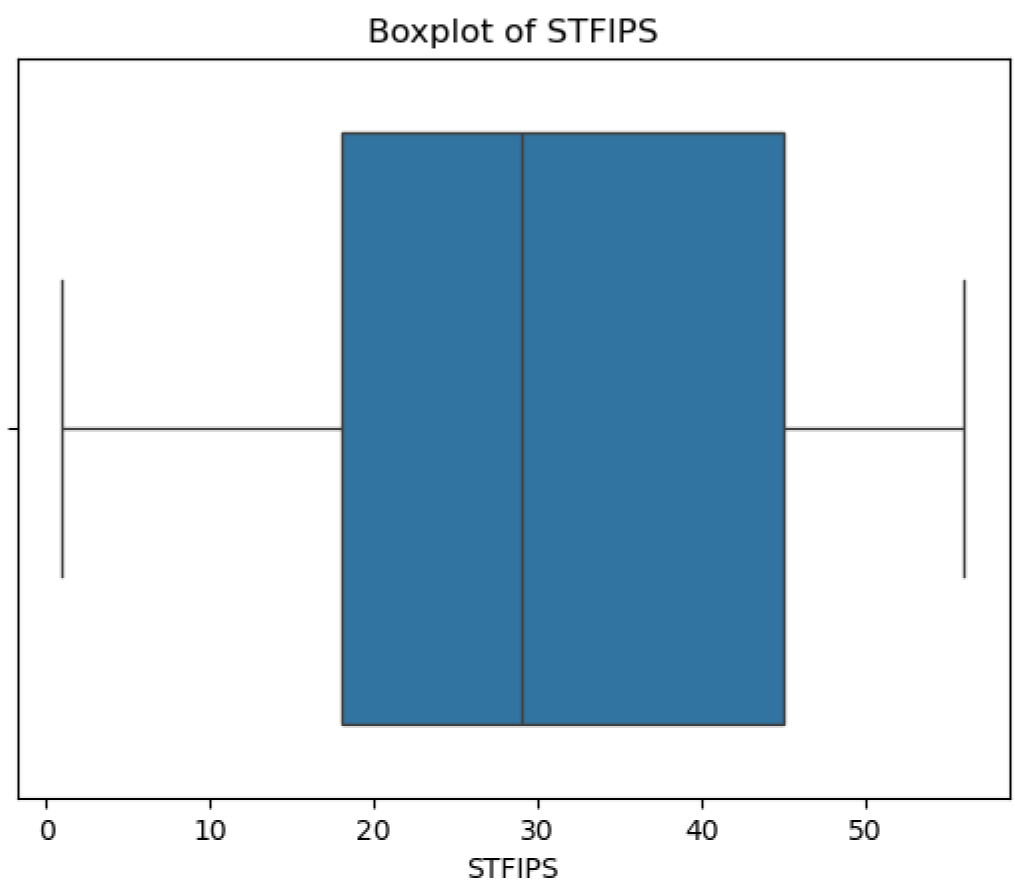
In [24]:

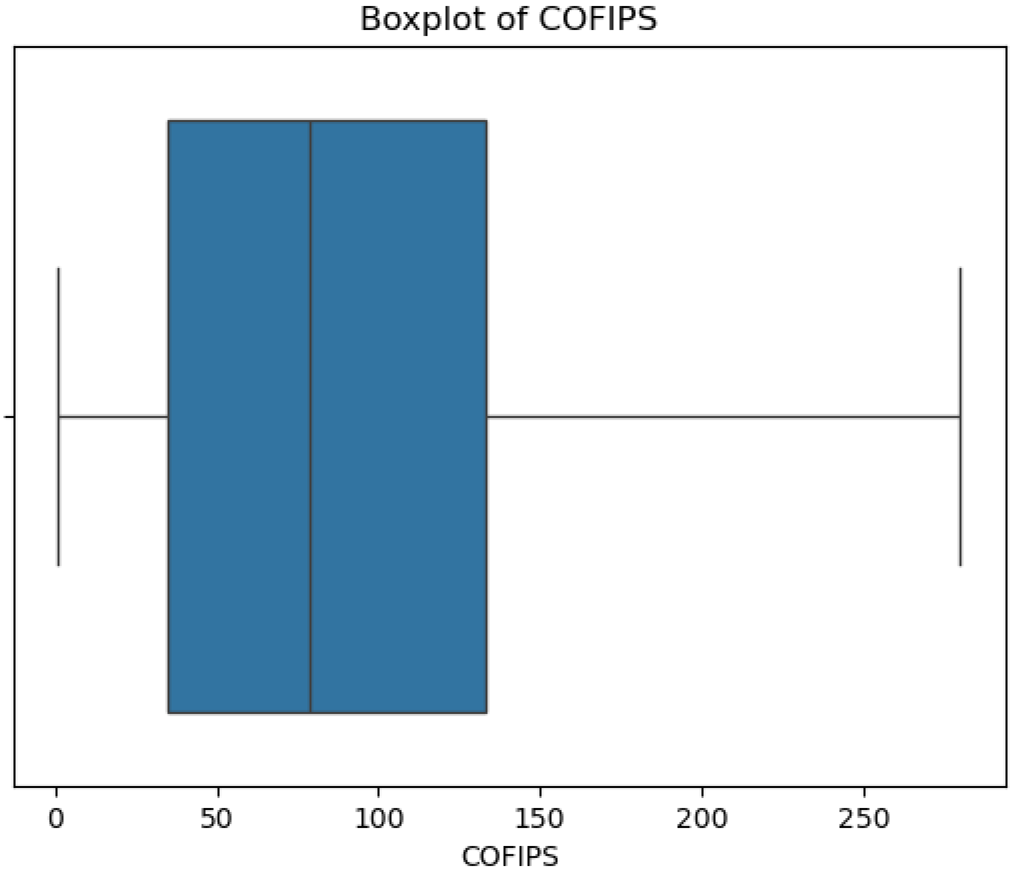
```
# creating custom definition to remove outliers using IQR method
def outliers(data):
    for col in data.select_dtypes(include=['int64','float64']).columns:
        Q1 = data[col].quantile(0.25)
        Q3 = data[col].quantile(0.75)
        IQR = Q3 - Q1

        lower = Q1 - (1.5*IQR)
        upper = Q3 + (1.5*IQR)

        # Capping
        data[col] = data[col].apply(lambda x: lower if x < lower else upper if x > upper

    return data
```

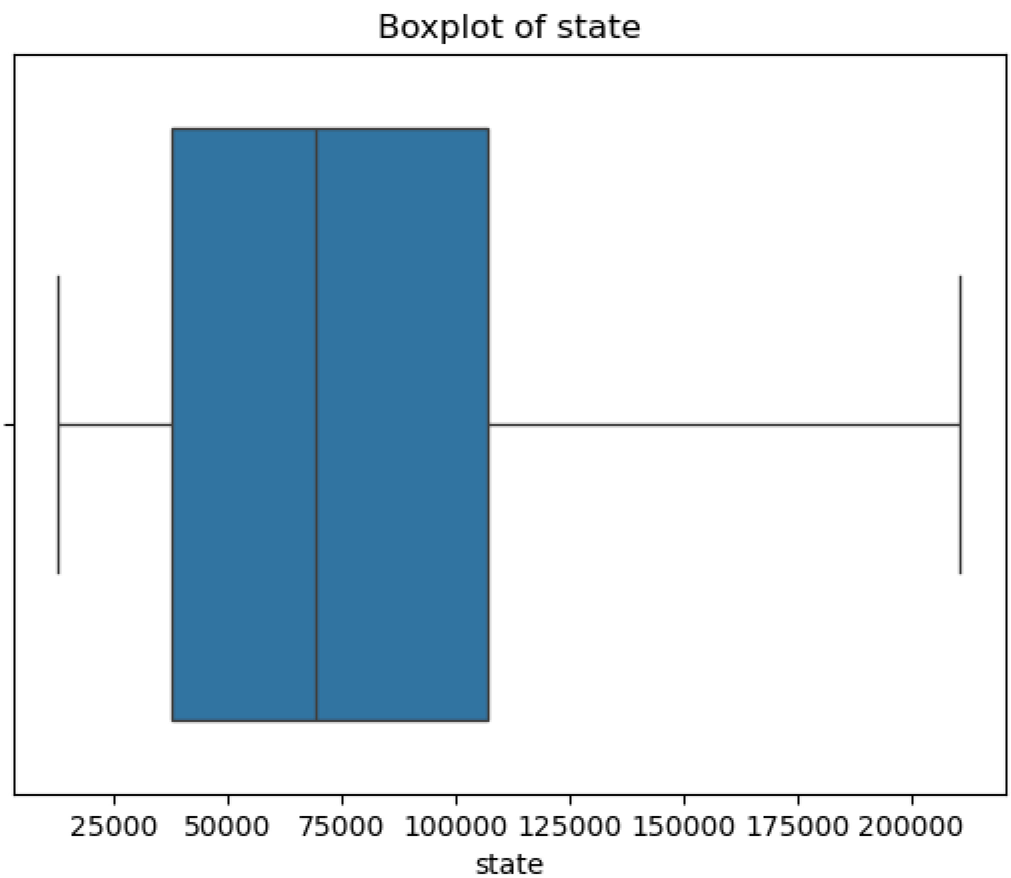
In [25]:

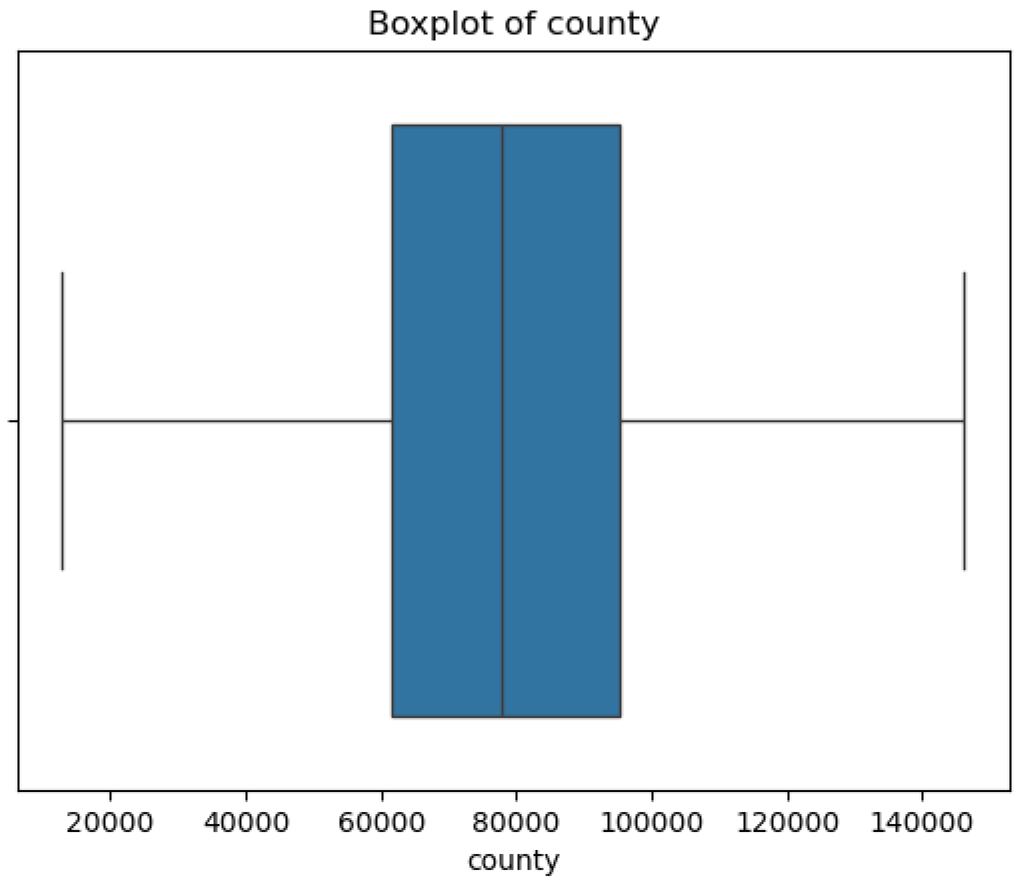
```
df4 = pd.DataFrame(df3)
outliers(df4)
### syntex to visualise the df to detect outliers for each columns
for col in df4.columns:
    sns.boxplot(data=df4,x=col)
    plt.title(f'Boxplot of {col}')
    plt.show()
```

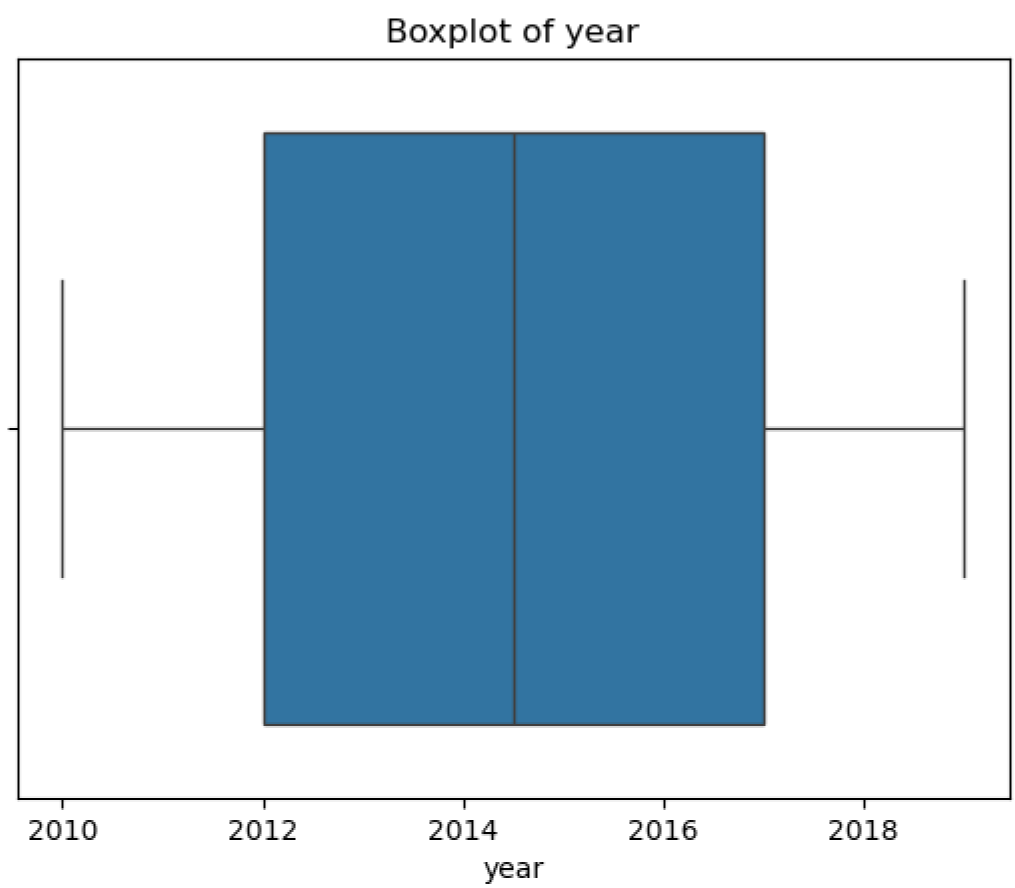

Boxplot of FIPS
0
10000
20000
30000
40000
50000
FIPS

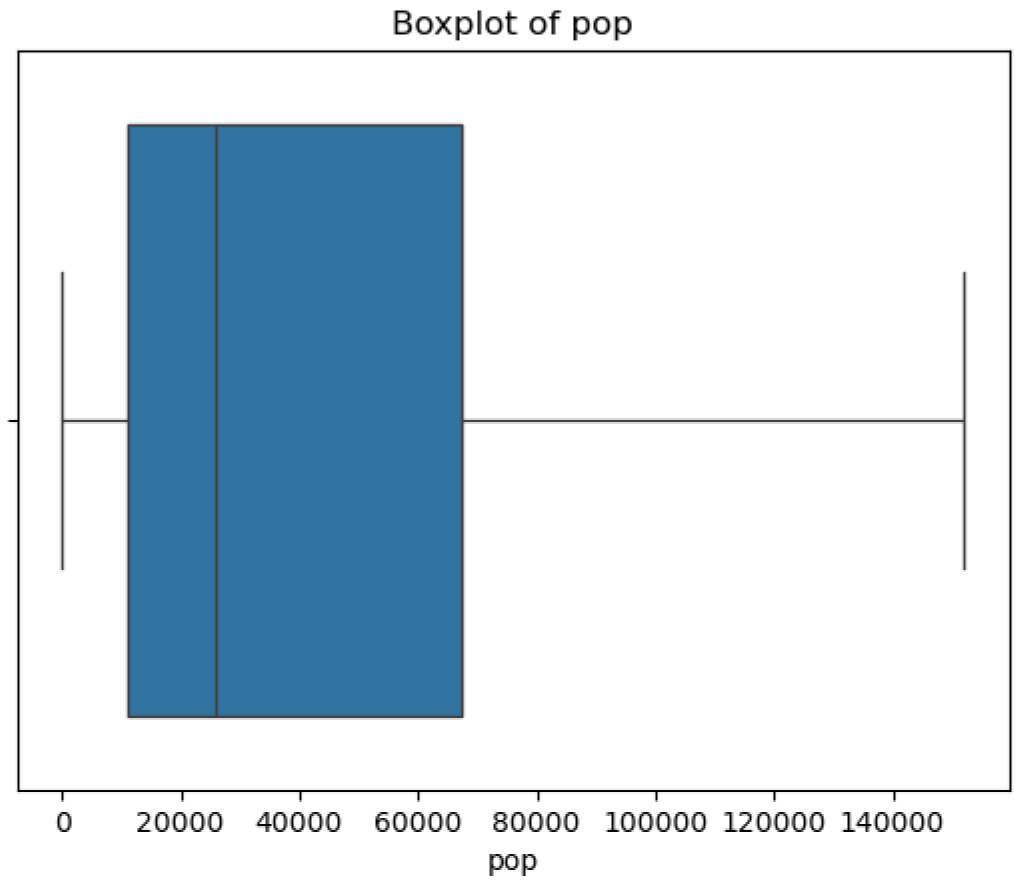

Boxplot of STFIPS
0
10
20
30
40
50
STFIPS

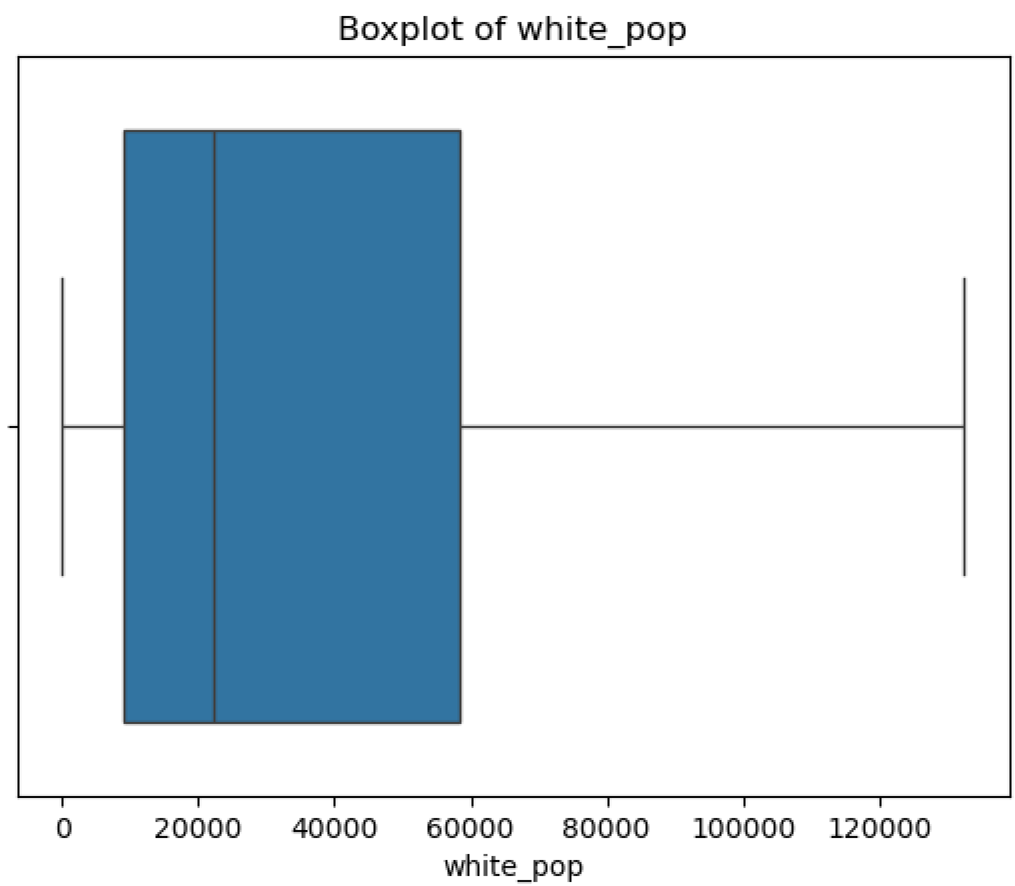

Boxplot of COFIPS
0
50
100
150
200
250
COFIPS

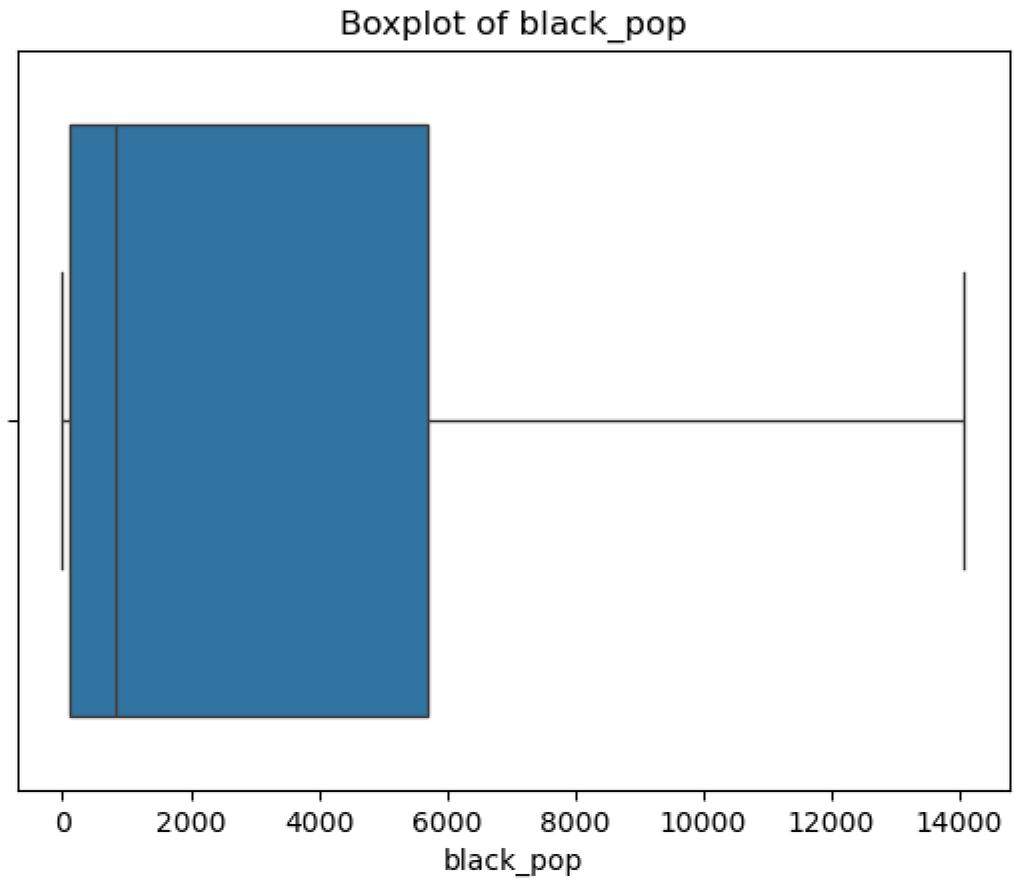

Boxplot of state
25000
50000
75000
100000
125000
150000
175000
200000
state

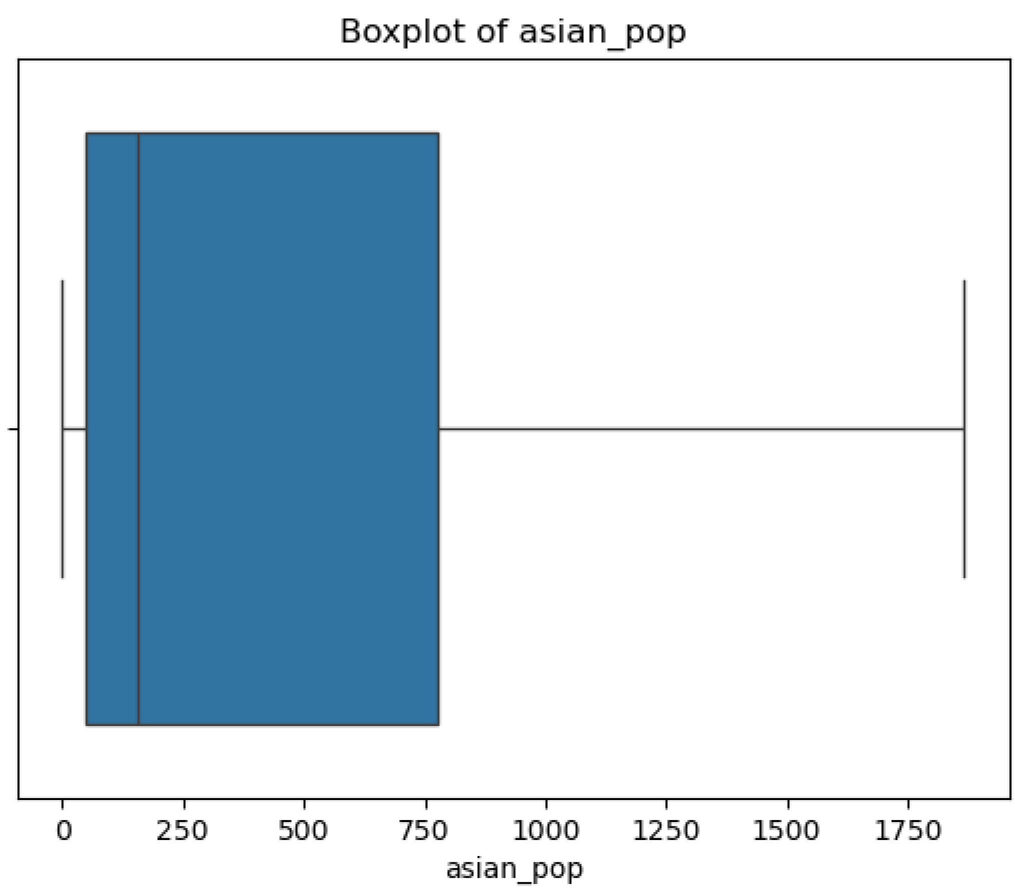

Boxplot of county
20000
40000
60000
80000
100000
120000
140000
county

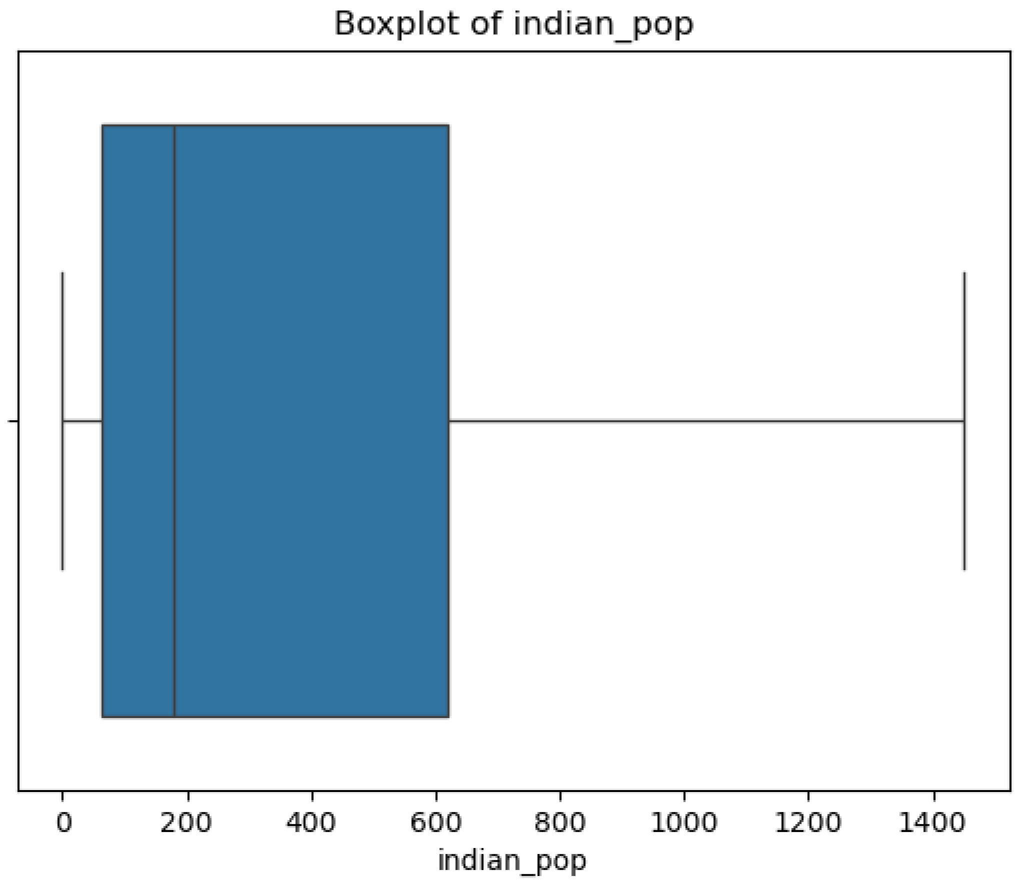

Boxplot of year
2010
2012
2014
2016
2018
year

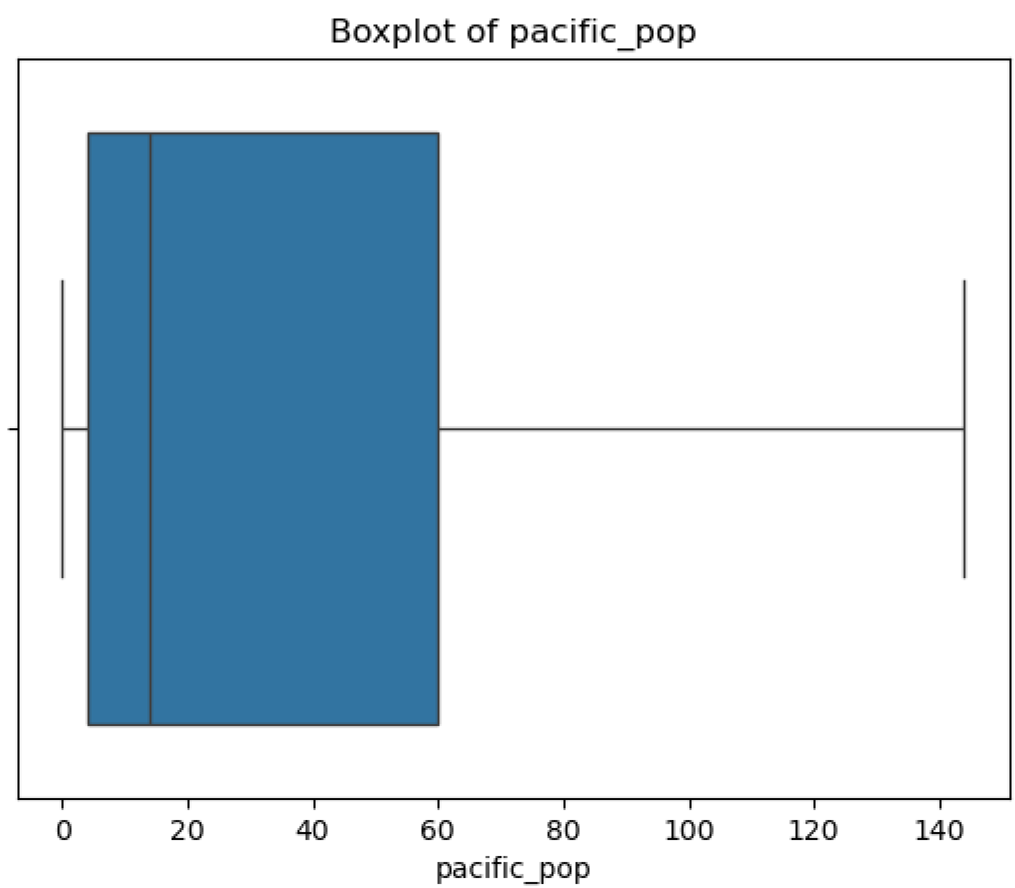

Boxplot of pop
0
20000
40000
60000
80000
100000
120000
140000
pop

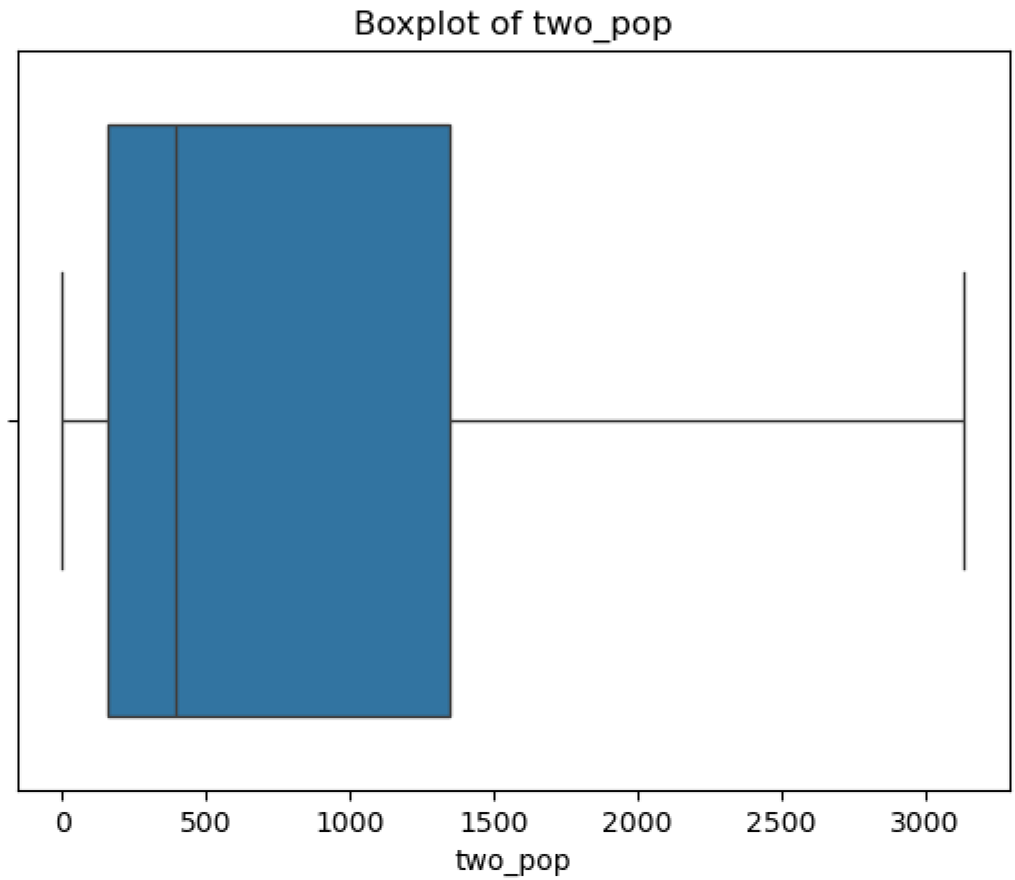

Boxplot of white_pop
0
20000
40000
60000
80000
100000
120000
white_pop

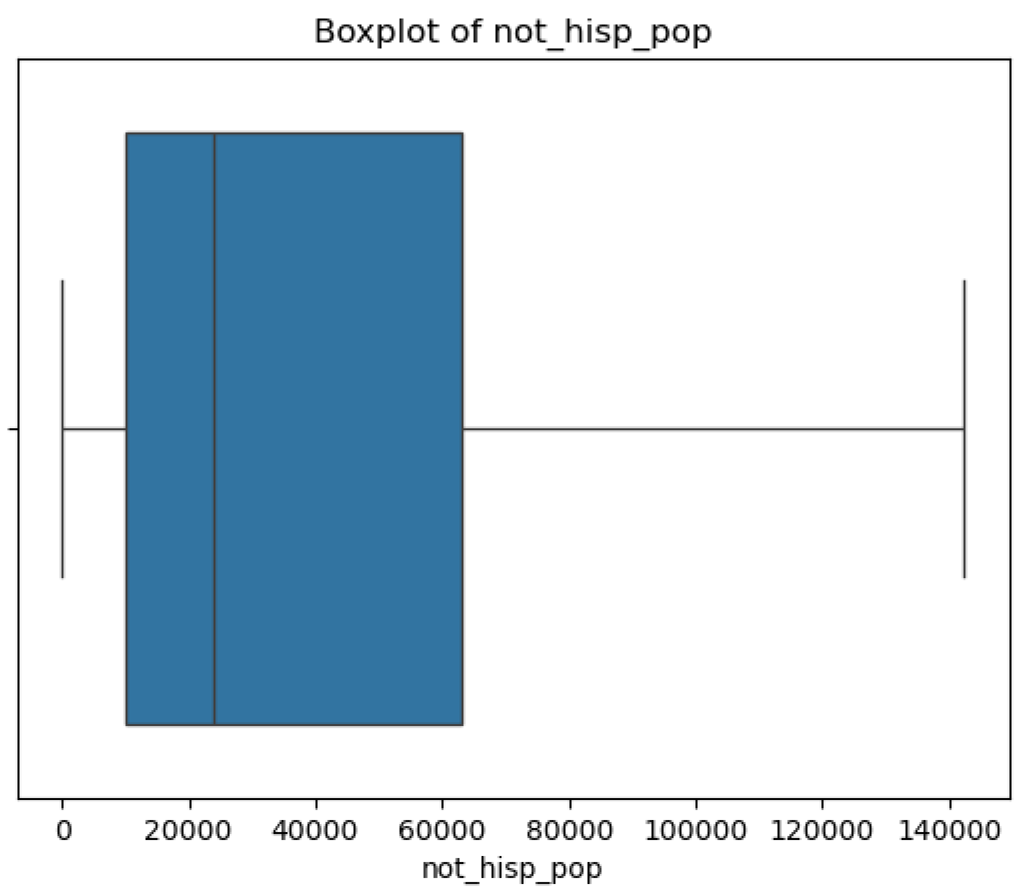

Boxplot of black_pop
0
2000
4000
6000
8000
10000
12000
14000
black_pop

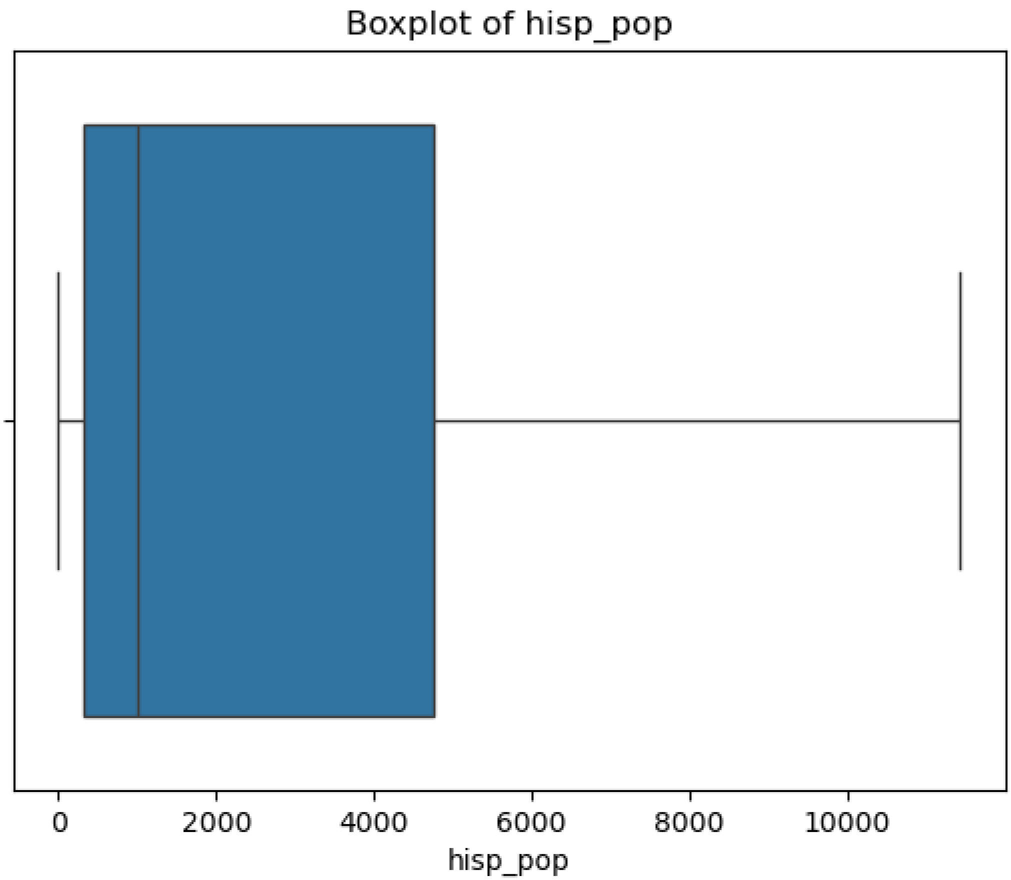

Boxplot of asian_pop
0
250
500
750
1000
1250
1500
1750
asian_pop


Boxplot of indian_pop
0
200
400
600
800
1000
1200
1400
indian_pop


Boxplot of pacific_pop
0
20
40
60
80
100
120
140
pacific_pop


Boxplot of two_pop
0
500
1000
1500
2000
2500
3000
two_pop


Boxplot of not_hisp_pop
0
20000
40000
60000
80000
100000
120000
140000
not_hisp_pop



In [26]:
```
df4
```

Out[26]:

	FIPS	STFIPS	COFIPS	state	county	year	pop	white_pop	black_pop	asian_pop	indi
0	1001	1	1.0	72309.697015	88962.379901	2010	54571.0	43297.0	9689.0	484.0	
1	1001	1	1.0	72309.697015	88962.379901	2011	55227.0	43699.0	9883.0	514.0	
2	1001	1	1.0	72309.697015	88962.379901	2012	54954.0	43315.0	9949.0	552.0	
3	1001	1	1.0	72309.697015	88962.379901	2013	54727.0	42943.0	9984.0	561.0	
4	1001	1	1.0	72309.697015	88962.379901	2014	54893.0	42945.0	10103.0	573.0	
...	...	...	...	...	...	...	...	...	...	...	
31405	56045	56	45.0	25117.400058	76044.504868	2015	7208.0	6835.0	39.0	81.0	
31406	56045	56	45.0	25117.400058	76044.504868	2016	7220.0	6826.0	38.0	88.0	
31407	56045	56	45.0	25117.400058	76044.504868	2017	6968.0	6558.0	44.0	97.0	
31408	56045	56	45.0	25117.400058	76044.504868	2018	6924.0	6474.0	47.0	109.0	
31409	56045	56	45.0	25117.400058	76044.504868	2019	6927.0	6454.0	48.0	117.0	

31410 rows × 15 columns

Feature Scaling

Min Max scaling

In [28]:
```
df5 = pd.DataFrame(df4)
```

In [29]:
```
scaling=MinMaxScaler()
numerical_col = df5.select_dtypes(include=['int64','float64'])
```

```
df5 = scaling.fit_transform(numerical_col)
df5=pd.DataFrame(df5, columns=numerical_col.columns, index=df1.index) #datafrme after mi
```

PowerTransformer

In [31]:
```
pt = PowerTransformer(method='yeo-johnson',standardize=True)
numeriacal_features = df5.select_dtypes(include=['int64','float64']).columns
df5[numeriacal_features] = pt.fit_transform(df5[numeriacal_features])
```

In [32]:
```
df5
```

Out[32]:

	FIPS	STFIPS	COFIPS	state	county	year	pop	white_pop	black_pop	asian_po
0	-1.928755	-1.92287	-1.726894	0.021723	0.285893	-1.610567	0.616111	0.527443	1.420259	0.56683
1	-1.928755	-1.92287	-1.726894	0.021723	0.285893	-1.230735	0.629712	0.537918	1.432322	0.62500
2	-1.928755	-1.92287	-1.726894	0.021723	0.285893	-0.860716	0.624072	0.527914	1.436322	0.69388
3	-1.928755	-1.92287	-1.726894	0.021723	0.285893	-0.499397	0.619361	0.518138	1.438423	0.70945
4	-1.928755	-1.92287	-1.726894	0.021723	0.285893	-0.145869	0.622808	0.518191	1.445458	0.72980
...	...	...	...	...	...	...	...	...	...	.
31405	1.699176	1.70619	-0.608586	-1.335590	-0.100617	0.200617	-1.082332	-1.018956	-0.965844	-0.71266
31406	1.699176	1.70619	-0.608586	-1.335590	-0.100617	0.540690	-1.081625	-1.019552	-0.966576	-0.67894
31407	1.699176	1.70619	-0.608586	-1.335590	-0.100617	0.874886	-1.096522	-1.037393	-0.962191	-0.63641
31408	1.699176	1.70619	-0.608586	-1.335590	-0.100617	1.203665	-1.099133	-1.043013	-0.960002	-0.58111
31409	1.699176	1.70619	-0.608586	-1.335590	-0.100617	1.527427	-1.098955	-1.044353	-0.959273	-0.54512

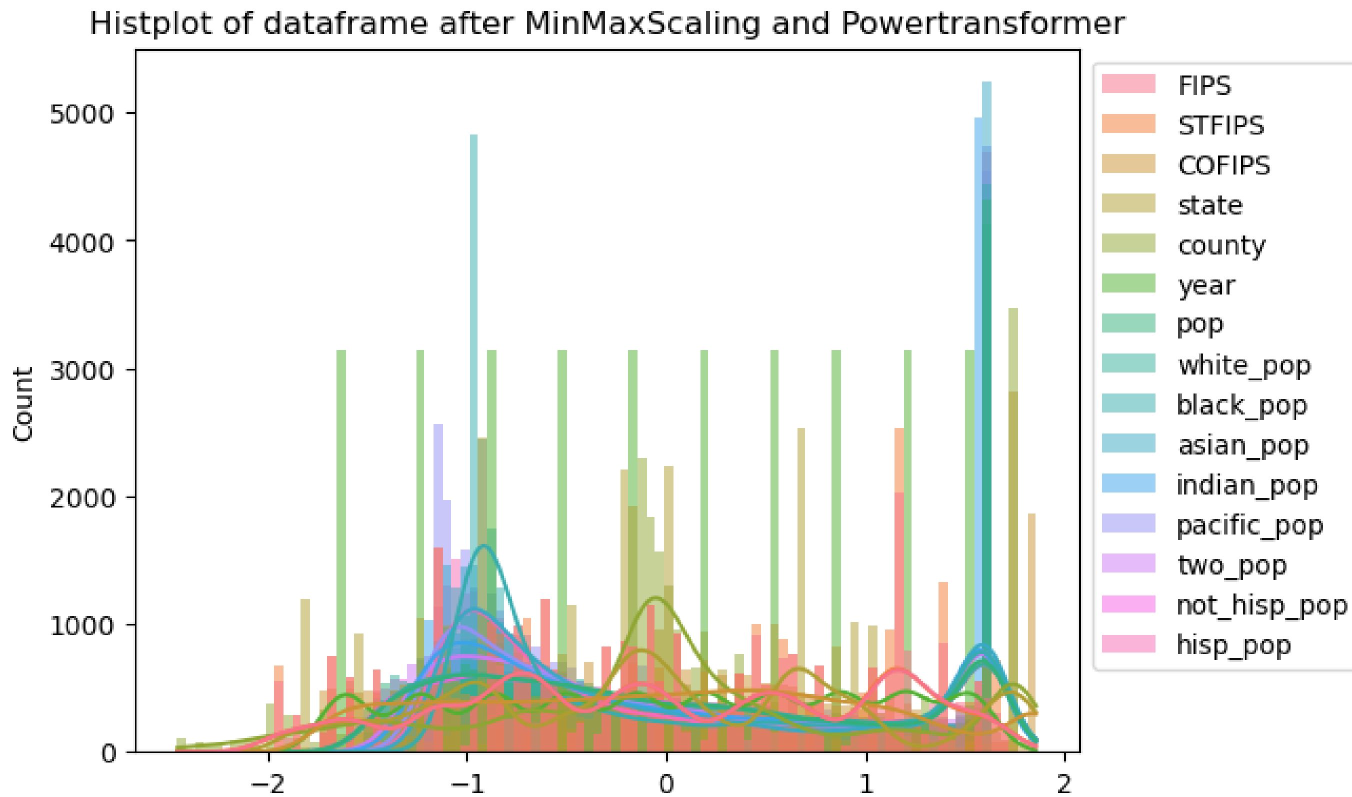
31410 rows × 15 columns

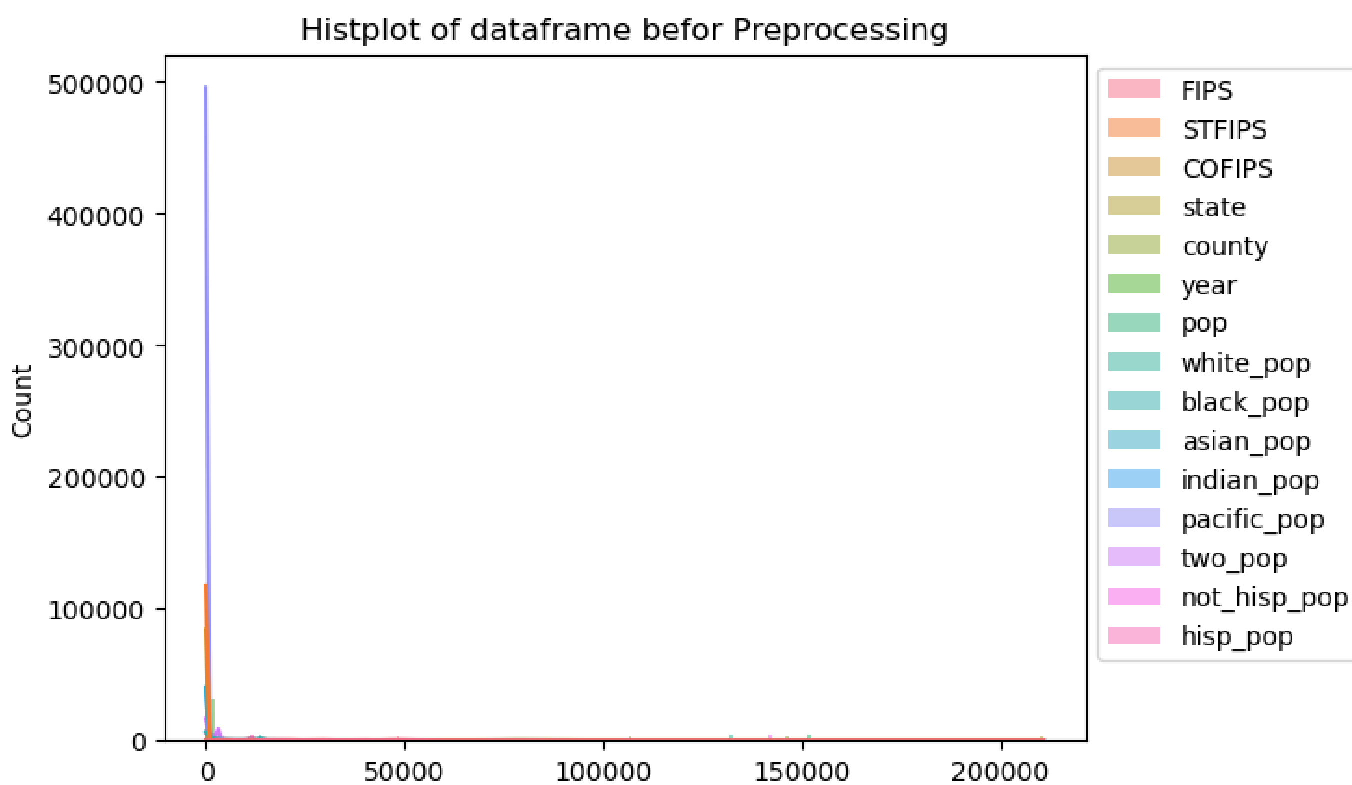
In [33]:
```
skewness = df5.skew()
print_section("\033[1mSkewness of Features in dataframe after Scaling\033[0m")
print_section(skewness)
```

```
------------------------------------------------------------
Skewness of Features in dataframe after Scaling
------------------------------------------------------------
------------------------------------------------------------
FIPS          -0.070539
STFIPS        -0.071084
COFIPS         0.098267
state          0.097310
county         0.000022
year          -0.066401
pop            0.336701
white_pop      0.336056
black_pop      0.557582
asian_pop      0.554942
indian_pop     0.438283
pacific_pop    0.486517
two_pop        0.444147
not_hisp_pop   0.335262
hisp_pop       0.505745
dtype: float64
------------------------------------------------------------
```

In [34]:
```
ax=sns.histplot(df5,kde=True,linewidth=0,legend=True)
plt.title('Histplot of dataframe after MinMaxScaling and Powertransformer')
sns.move_legend(ax, "upper left", bbox_to_anchor=(1, 1))
```

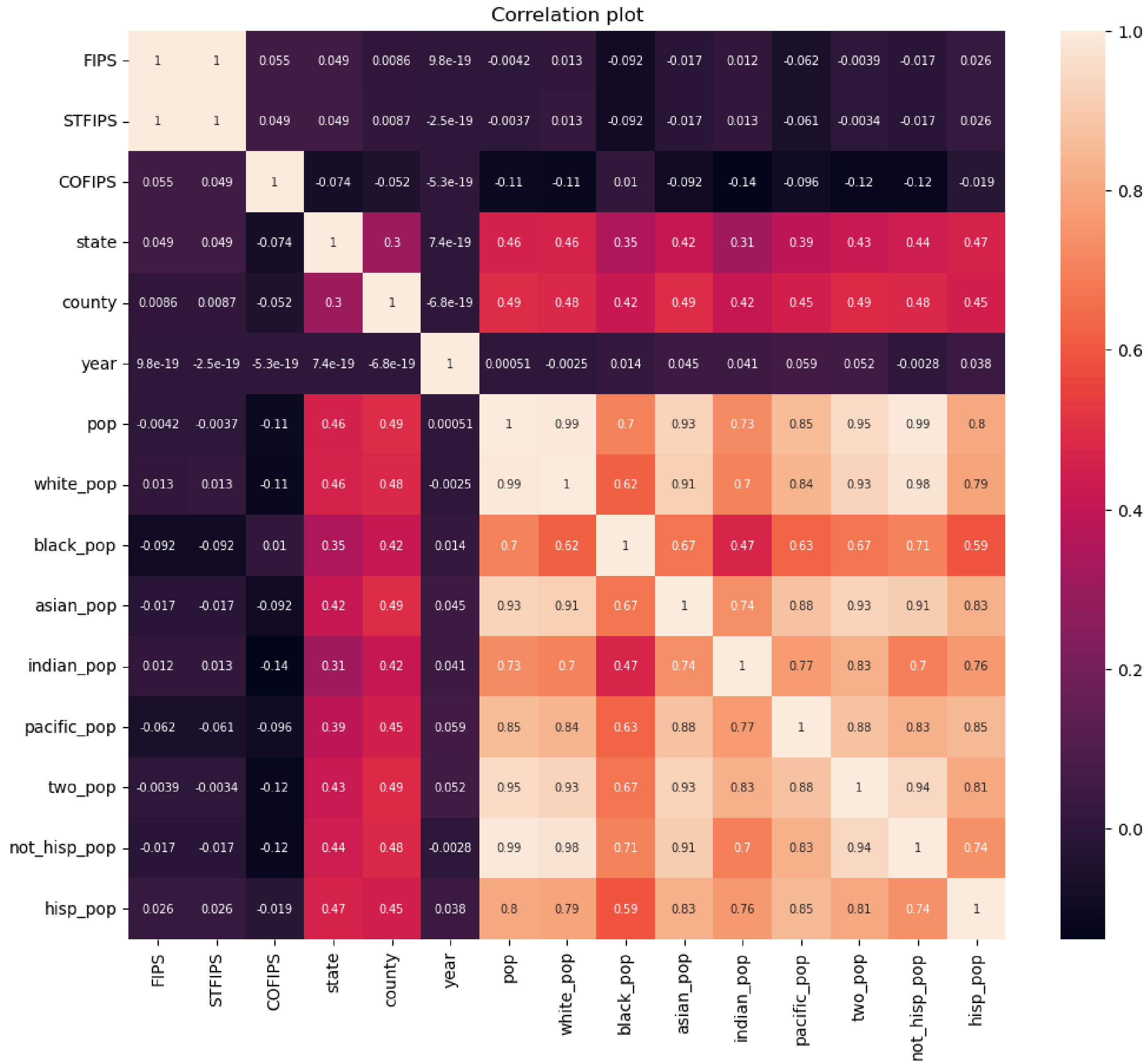
```
plt.show()

ax=sns.histplot(df4,kde=True,linewidth=0,legend=True)
plt.title('Histplot of dataframe befor Preprocessing')
sns.move_legend(ax, "upper left", bbox_to_anchor=(1, 1))
plt.show()
```





In [35]:
```
#correlation of data frame after EDA
cor_df = df5.corr()
#correlation plot
plt.figure(figsize=(12,10))
sns.heatmap(cor_df,annot=True,annot_kws={'size': 7})
```

```
plt.title('Correlation plot')
plt.show()
```



Feature Selection

In [37]:
```
#converting df5 to new dataset name for futher process
df_pop = pd.DataFrame(df5)
```

In [38]:
```
x = df_pop.drop('pop', axis=1)
y = df_pop['pop']
```

In [39]:
```
sk = SelectKBest(score_func=f_regression,k=14)
x_new = sk.fit_transform(x,y)
```

In [40]:
```
#Get selected feture names and scores
selected_features = x.columns[sk.get_support()]
features_scores = pd.DataFrame({'feature':x.columns,'Score':sk.scores_}).sort_values(by=

print_title('Selected Features:')
print_section(pd.DataFrame(list(selected_features)))
print_title("\nFeature Scores:")
print_section(features_scores)
```

```
---------------------------------------------------------------
Selected Features:
---------------------------------------------------------------
              0
0          FIPS
1        STFIPS
2        COFIPS
3         state
4        county
5          year
6     white_pop
7     black_pop
8     asian_pop
9    indian_pop
10  pacific_pop
11      two_pop
12 not_hisp_pop
13     hisp_pop
---------------------------------------------------------------

---------------------------------------------------------------

Feature Scores:
---------------------------------------------------------------
         feature         Score
12  not_hisp_pop  1.540729e+06
6      white_pop  1.164183e+06
11       two_pop  2.898884e+05
8      asian_pop  1.947178e+05
10   pacific_pop  8.474241e+04
13      hisp_pop  5.658111e+04
9     indian_pop  3.518288e+04
7      black_pop  3.100110e+04
4         county  9.930721e+03
3          state  8.474266e+03
2         COFIPS  3.584230e+02
0           FIPS  5.526506e-01
1         STFIPS  4.202460e-01
5           year  8.231369e-03
---------------------------------------------------------------
```

In [41]:
```
x_select=x[selected_features]
```

In [42]:
```
x_select.columns
```

Out[42]:
```
Index(['FIPS', 'STFIPS', 'COFIPS', 'state', 'county', 'year', 'white_pop',
       'black_pop', 'asian_pop', 'indian_pop', 'pacific_pop', 'two_pop',
       'not_hisp_pop', 'hisp_pop'],
      dtype='object')
```

In [43]:
```
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x_select)
```

In [44]:
```
# Split data into training and testing sets.
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42
```

In [45]:
```
print(f"""
shape x_train: {x_train.shape}
shape x_test: {x_test.shape}
shape y_train: {y_train.shape}
shape y_test: {y_test.shape}""")
```

```
shape x_train: (25128, 14)
shape x_test: (6282, 14)
```

```
shape y_train: (25128,)
shape y_test: (6282,)
```

Model selection

In [47]:
```
models = {
    '1.linear Regression':LinearRegression(),
    '2.Dicision Tree Regression':DecisionTreeRegressor(),
    '3.Random Forest Regressor':RandomForestRegressor(),
    '4.Gradient Boosting Regressor':GradientBoostingRegressor(),
    '5.Support Vector Regressor':SVR()
}
```

In [48]:
```
result = {}
for model_name, model in models.items():
    model.fit(x_train,y_train)
    y_pred = model.predict(x_test)
    mae = mean_absolute_error(y_test,y_pred)
    mse = mean_squared_error(y_test,y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test,y_pred)
    result[model_name] = {'mae':mae, 'mse':mse,'rmse':rmse,'r2':r2}
```

In [49]:
```
result_df = pd.DataFrame(result).T
print_title('Score details for variuse modles')
print_section(result_df)
```

```
------------------------------------------------------------
Score details for variuse modles
------------------------------------------------------------
                                   mae       mse      rmse        r2
1.linear Regression           0.041862  0.005841  0.076427  0.994166
2.Dicision Tree Regression    0.004371  0.000115  0.010710  0.999885
3.Random Forest Regressor     0.002214  0.000040  0.006327  0.999960
4.Gradient Boosting Regressor 0.017837  0.000737  0.027143  0.999264
5.Support Vector Regressor    0.043591  0.002703  0.051986  0.997301
------------------------------------------------------------
```

Hyperparameter turning

In [371…
```
rfg = RandomForestRegressor(random_state=42,
                            n_estimators=50,
                            max_depth=10,
                            min_samples_split=5,
                            min_samples_leaf=2,
                            max_features='sqrt',
                            ccp_alpha=0.1,
                            n_jobs=-1)
```

In [373…
```
param_grid = {
    'n_estimators':[50,100,200,300],
    'max_depth':[10,20,None],
    'min_samples_split':[2,5,10],
    'min_samples_leaf':[1,2,4]
}
```

In [375…
```
RandomGrid = RandomizedSearchCV(estimator = rfg,
                                param_distributions=param_grid,
                                cv=10,
                                scoring = 'r2',
                                n_jobs=-1,
                                verbose=2)
```

In [377…

```
RandomGrid.fit(x_train,y_train)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
```

Out[377]:

RandomizedSearchCV

best_estimator_: RandomForestRegressor

RandomForestRegressor

In [379…

```
print("Best Parameters:",RandomGrid.best_params_)
print("best R2 Score:",RandomGrid.best_score_)
```

```
Best Parameters: {'n_estimators': 50, 'min_samples_split': 5, 'min_samples_leaf': 1, 'ma
x_depth': 10}
best R2 Score: 0.8899507162245073
```