Knowledge Discovery & Data Mining

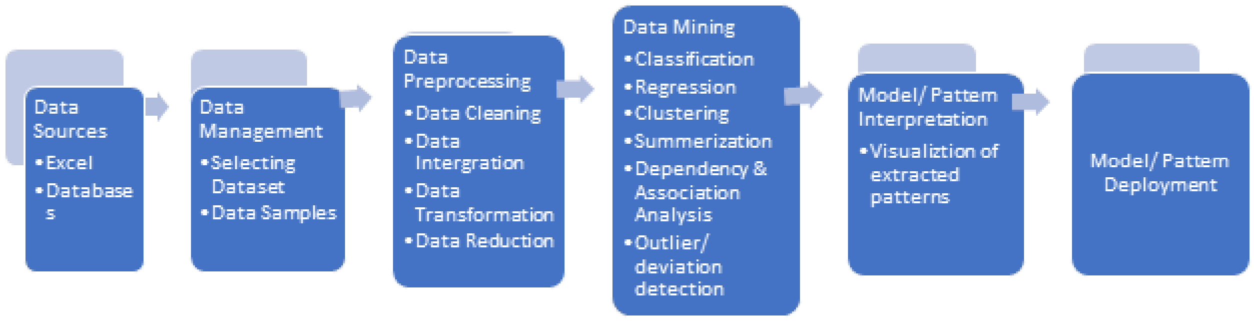
(Report by Bhoomi Kalpesh Patel and Srinarayan Srikanthan)

Problem I. Knowledge Discovery in Databases

1. Knowledge discovery in a database

It is the process of mining the abundantly available data in-order to look for and find interesting trends or patterns within data.

2.Steps of knowledge discovery



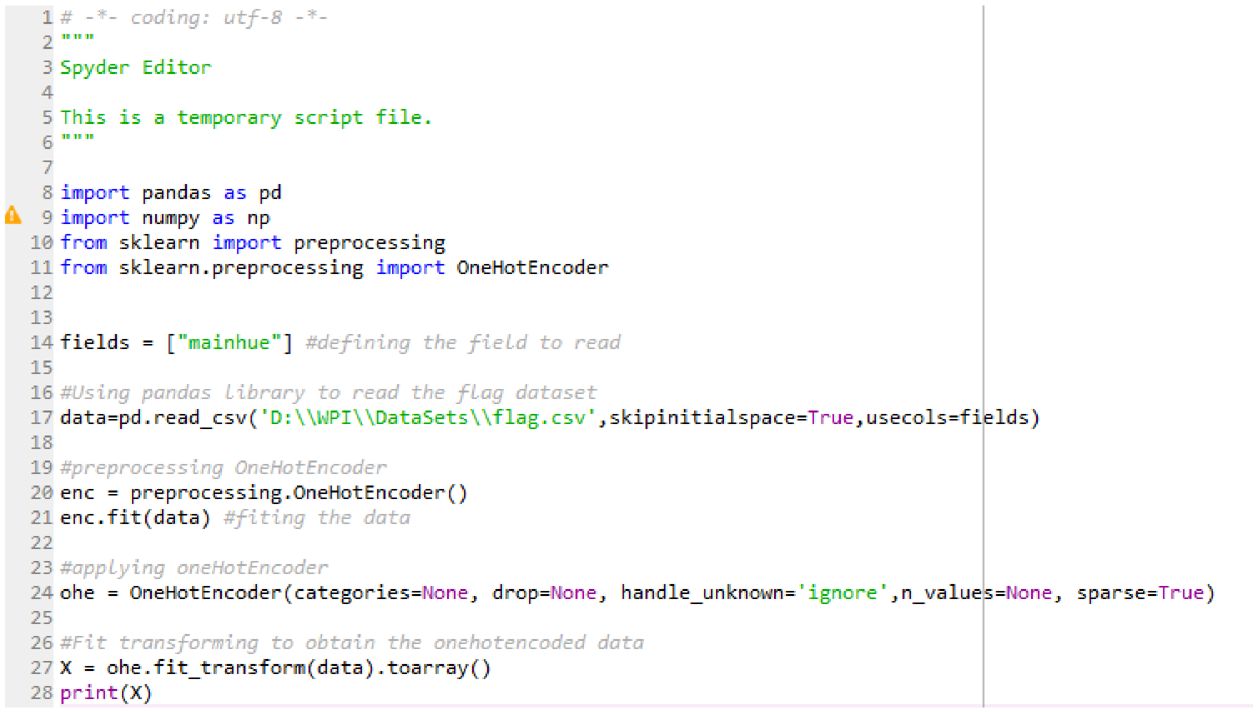
3.Data mining

Data Mining is a sub-process involved in KDD, which applies various types of algorithms such as Classification, Regression, Clustering, etc. to generate patterns which can help in predicting useful information.

Problem II. Data Preprocessing: Attribute Transformation

1. **Discrete attributes with too many values:**
   Even though, the attribute "name" contains many values, it is an integral part of the data and cannot be removed from the dataset. This is because when the patterns are generated, it will help us in identifying each data instances individually. However, while performing data preprocessing operations, we need not consider this attribute and use it as an index to map the data instances.

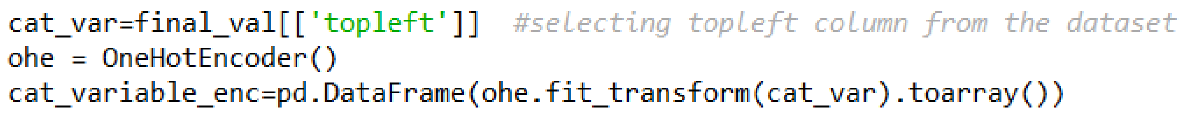
2. **Converting discrete attributes to continuous:**
   1) **OneHotEncode on *mainhue, topleft & botright***
      1. ***Mainhue***



```python
# -*- coding: utf-8 -*-
"""
Spyder Editor

This is a temporary script file.
"""

import pandas as pd
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder


fields = ["mainhue"] #defining the field to read

#Using pandas library to read the flag dataset
data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

#preprocessing OneHotEncoder
enc = preprocessing.OneHotEncoder()
enc.fit(data) #fiting the data

#applying oneHotEncoder
ohe = OneHotEncoder(categories=None, drop=None, handle_unknown='ignore',n_values=None, sparse=True)

#Fit transforming to obtain the onehotencoded data
X = ohe.fit_transform(data).toarray()
print(X)
```

Fig 2.1.1 Code to OneHotEncode Mainhue

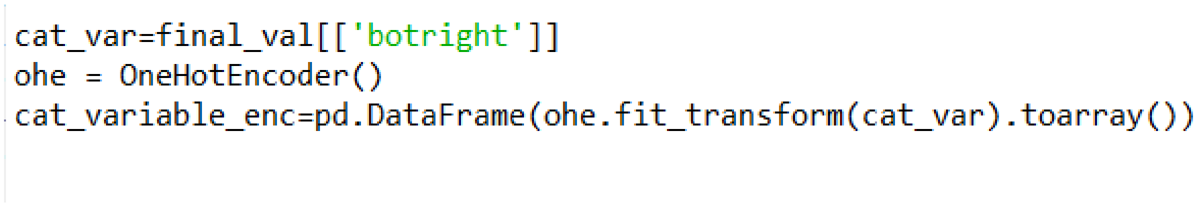
2. ***Topleft***



```
cat_var=final_val[['topleft']]  #selecting topleft column from the dataset
ohe = OneHotEncoder()
cat_variable_enc=pd.DataFrame(ohe.fit_transform(cat_var).toarray())
```

Fig 2.1.2 Code to OneHotEncode Topleft

3. ***Botright***



```
cat_var=final_val[['botright']]
ohe = OneHotEncoder()
cat_variable_enc=pd.DataFrame(ohe.fit_transform(cat_var).toarray())
```

Fig 2.1.3 Code to OneHotEncode Botright

2) **Attributes *landmass, zone, language and religion***

a) For each of the above mentioned attributes, discrete values are used which is inappropriate

Following are the reasons:

- Nominal data is only used as a label
- They do not represent any specific meaning to the value it bears
- Hence, it becomes ambiguous and fails to provide any meaningful information
- For example, in the given dataset, the country India belongs to Landmass 5, Zone 1, speaks language 684 and has religion 6. The above line is vague and we cannot interpret any useful information out of it.
- Moreover, using numbers to represent nominal variables brings in precedence among the values. For example when NE-1 and NW-2, it does not imply NW is greater than NE.

b) Using OneHotEncoder for the above mentioned attributes

i) It is better-off to use drop='first' instead of the default value which is None.By this we can reduce a column for a category,which can be inferred from the remaining data

```
fields = ["landmass","zone","language","religion"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)
enc = preprocessing.OneHotEncoder()

enc.fit(data)
ohe = OneHotEncoder(categories='auto', drop='first', handle_unknown='error',sparse=True)

X = ohe.fit_transform(data).toarray()
print(X)
```

Fig 2.2.1 OneHotEncoding using *drop* parameter

ii) The parameters used along with OneHotEncoder are as follows:
Categories:which by default is auto, but we can also specify the category expected at a particular column.
Drop:it specifies the categories that has to be dropped,which can be an array or a list.
Handle_unknown: it either raises an error or ignores when an unknown category is being encountered. If drop parameter is passed handle_unknown should be set to error.
Sparse: returns a sparse matrix or an array based on the boolean input.

3. Handling Missing Values

1. Original Graph

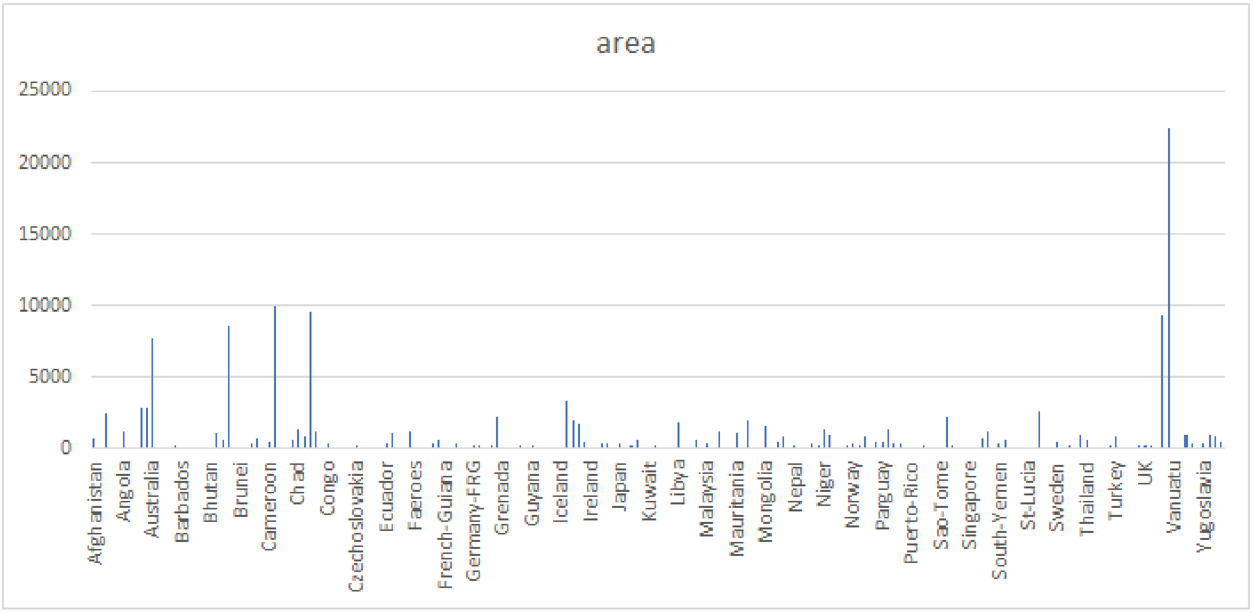
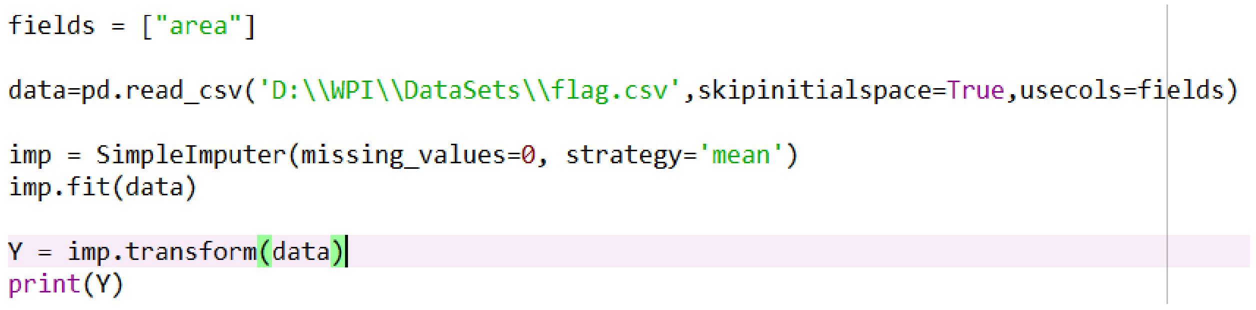


Fig 3.1 Original graph of area column with missing values

2. Univariate feature imputation using SimpleImputer

- **Using Mean**



```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

imp = SimpleImputer(missing_values=0, strategy='mean')
imp.fit(data)

Y = imp.transform(data)
print(Y)
```

Fig 3.2 SimpleImputer code using mean as strategy

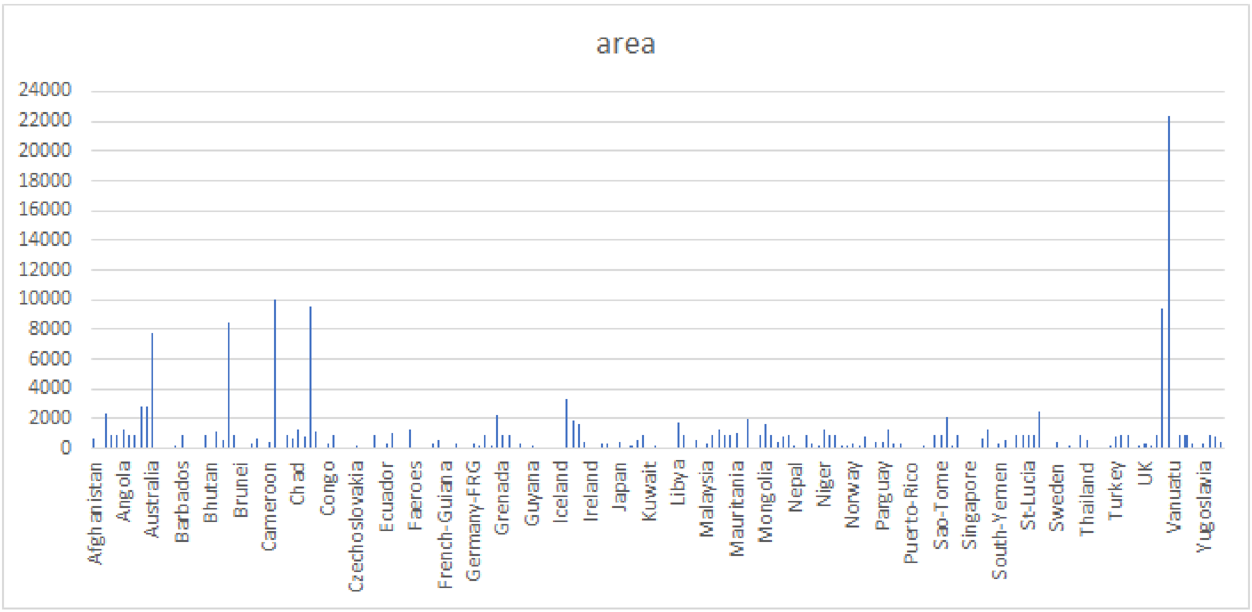
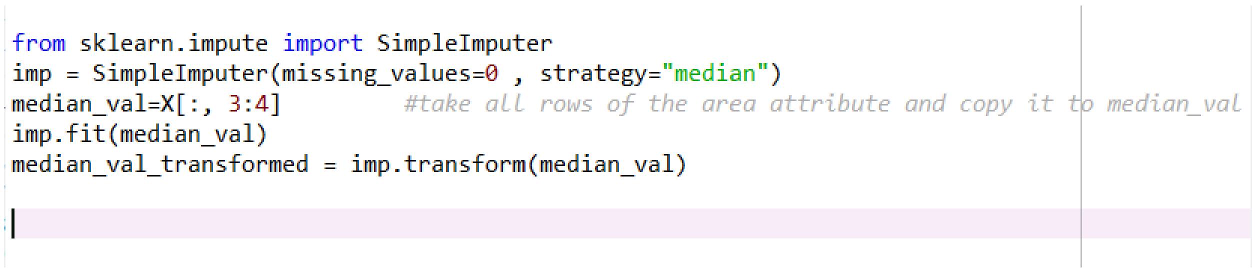


Fig. 3.3 Graph after using strategy as mean

This function calculates mean of all the values in the area (#4) column and replaces the missing value i.e 0 with the mean obtained. The mean obtained here was **848.806**

- **Using median**



```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=0 , strategy="median")
median_val=X[:, 3:4]          #take all rows of the area attribute and copy it to median_val
imp.fit(median_val)
median_val_transformed = imp.transform(median_val)
```

Fig 3.4 SimpleImputer code using median as strategy

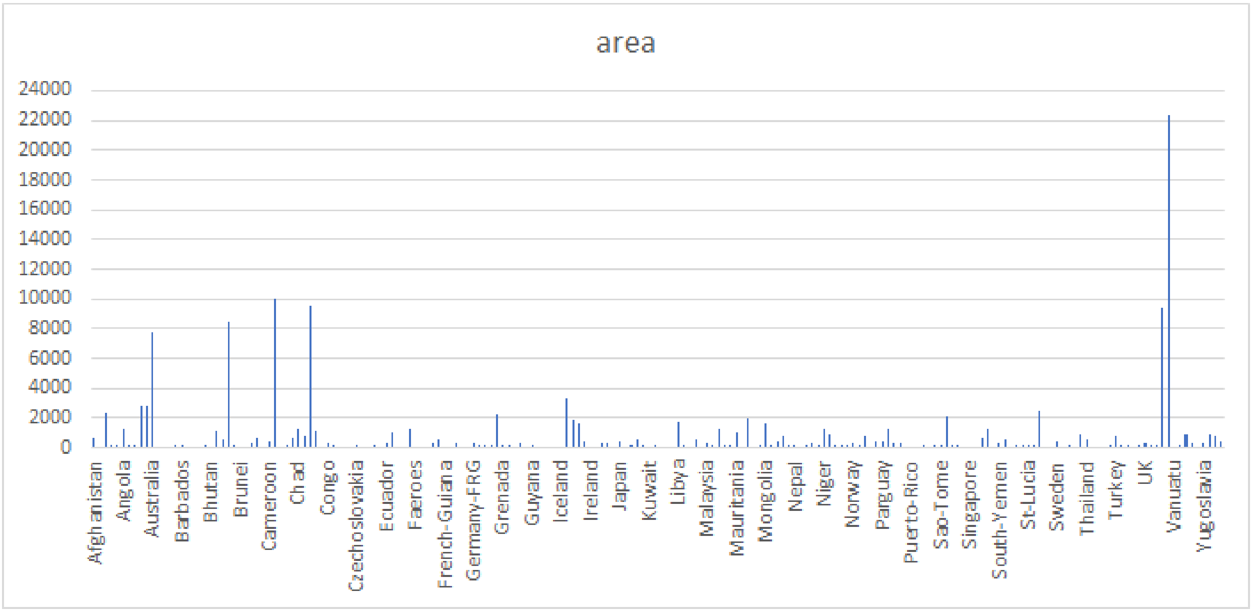
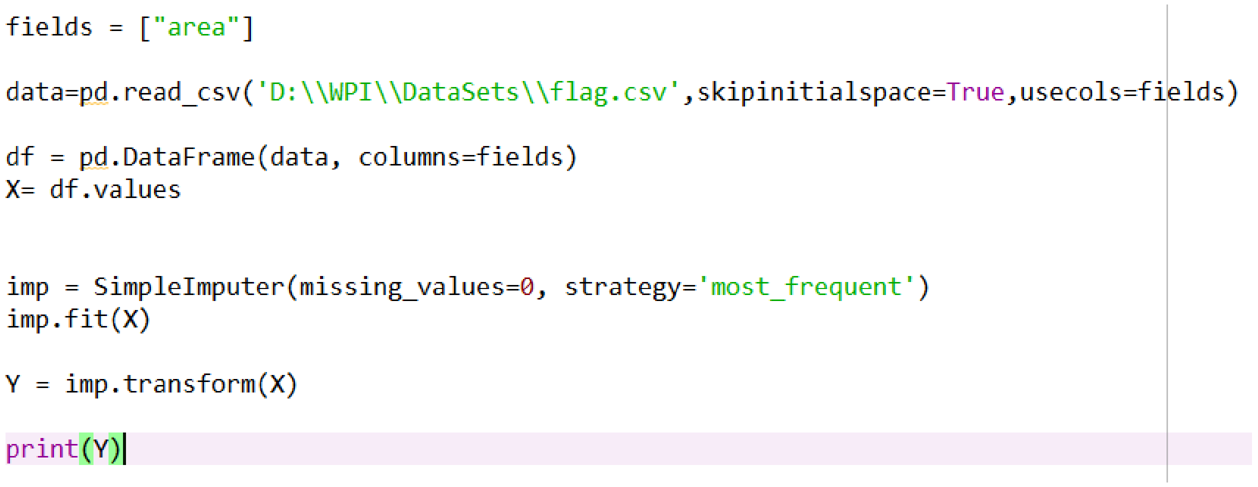


Fig 3.5 Graph after using strategy as median

This function calculates median of the all the values in the area (#4) column and replaces the missing value i.e 0 with the median obtained. The median obtained over here was **204**

- **Using most frequent**



```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)
X= df.values


imp = SimpleImputer(missing_values=0, strategy='most_frequent')
imp.fit(X)

Y = imp.transform(X)

print(Y)
```

Fig 3.6 SimpleImputer code using most_frequent as strategy

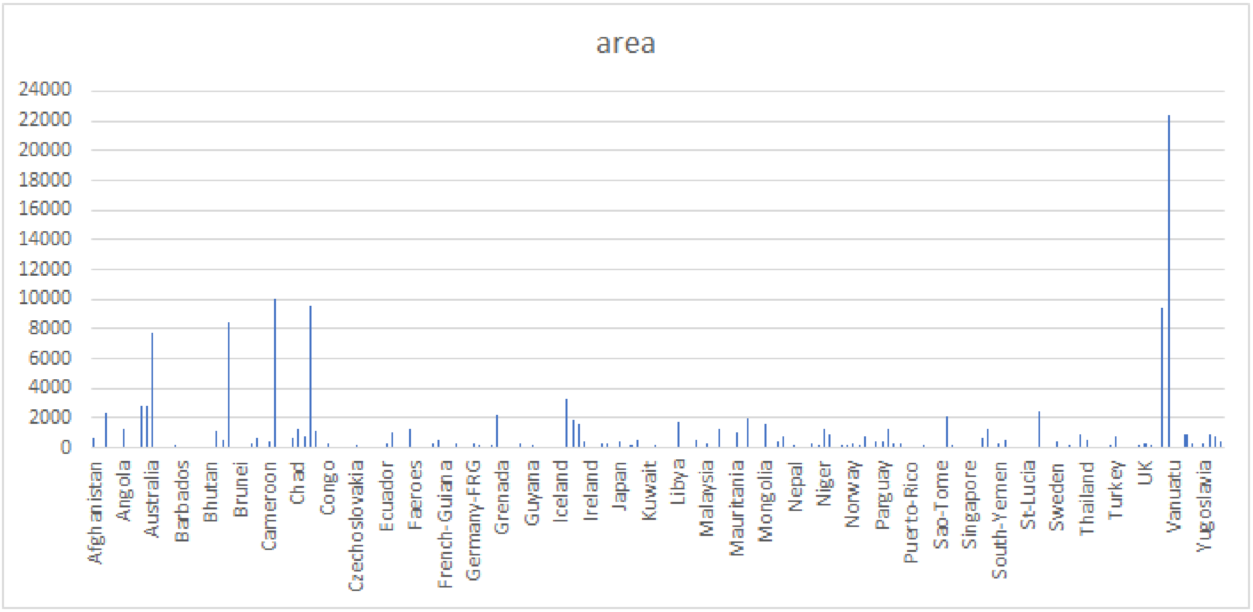
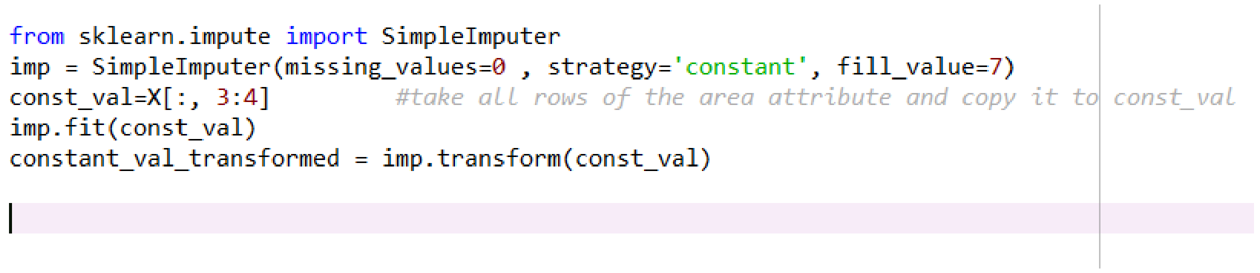


Fig 3.7 Graph after using strategy as most_frequent

This function finds the most occurring value in the column area (#4) and replaces the missing values with it. Here, the most frequent value obtained was **1**

- **Using constant**



```python
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=0 , strategy='constant', fill_value=7)
const_val=X[:, 3:4]         #take all rows of the area attribute and copy it to const_val
imp.fit(const_val)
constant_val_transformed = imp.transform(const_val)
```

Fig 3.8 SimpleImputer code using constant as strategy

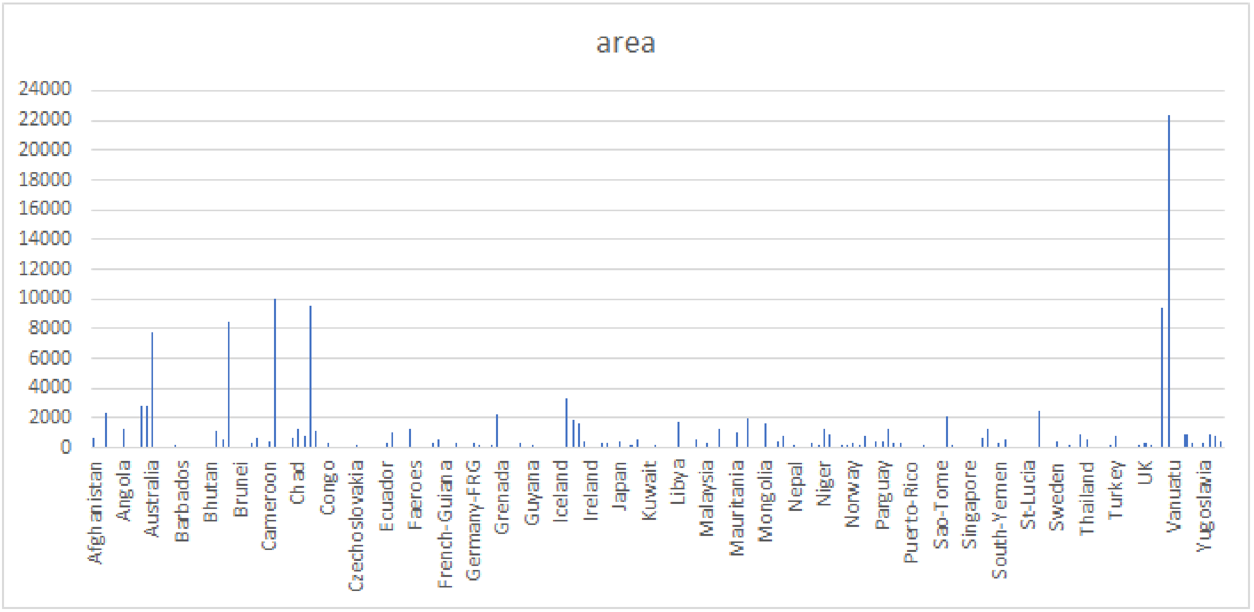
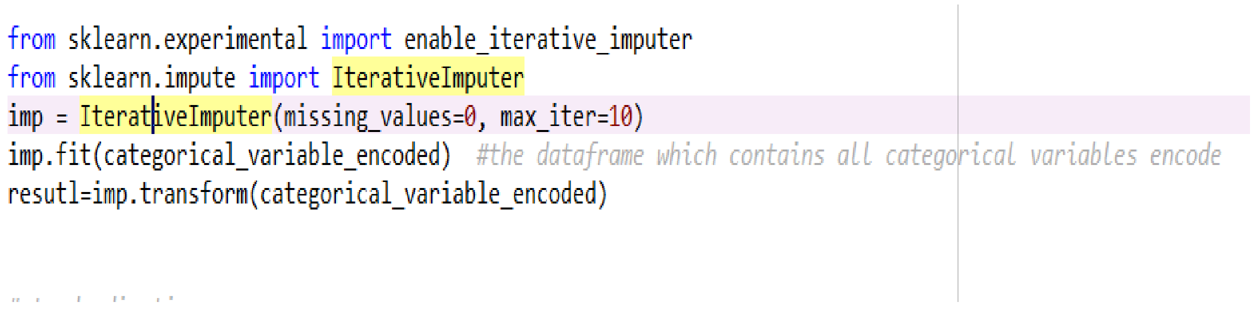


Fig 3.9 Graph after using strategy as constant

This function replaces all the missing value i.e 0 in the column area (#4) with the fill_value mentioned in the parameters. The default value is 0. In this case, the fill_value used was **7**

3. Multivariate feature imputation



```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
imp = IterativeImputer(missing_values=0, max_iter=10)
imp.fit(categorical_variable_encoded)  #the dataframe which contains all categorical variables encode
resutl=imp.transform(categorical_variable_encoded)
```

Fig 3.10 IterativeImputer code to handle missing data

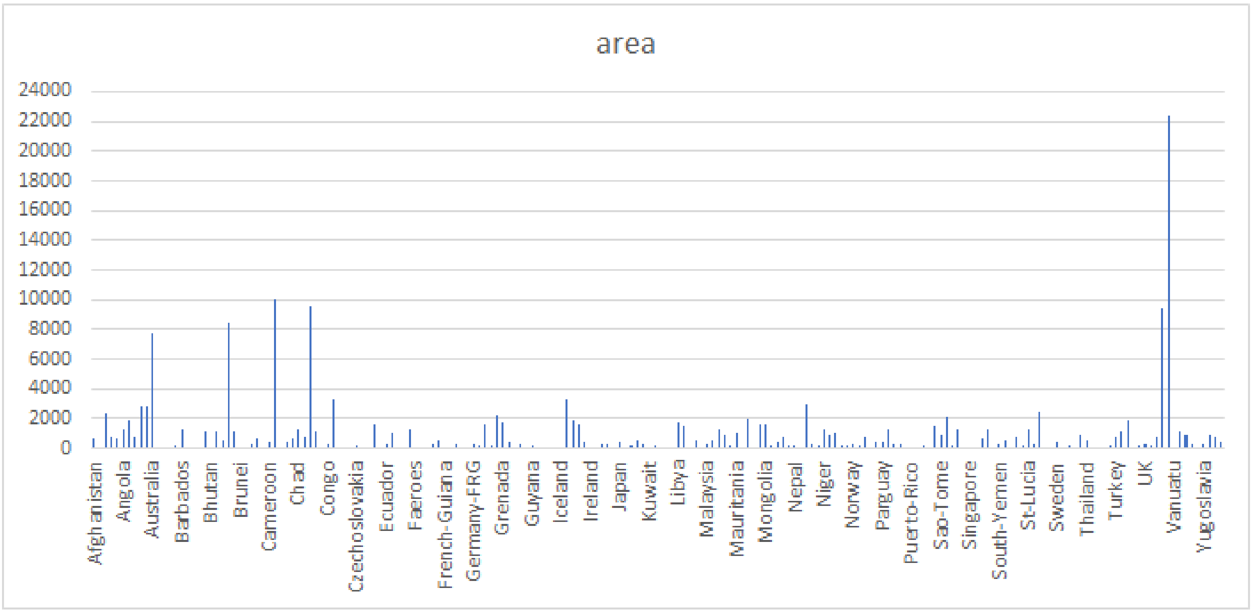


Fig 3.10 Graph after using IterativeImputer

This feature replaces the missing value i.e 0 in the column area (#4) with the help of each of the features available. In this case, the features that were considered are **landmass,zone,area,population,language,religion,bars,stripes,colours,red,green,blue,gold, white,black,orange,circles,crosses,saltries,quarters,sunstars,crescent,triangle,icon,animate, text**

4. **Standardization, scaling and normalization of continuous attributes**
    - **Original Graph**

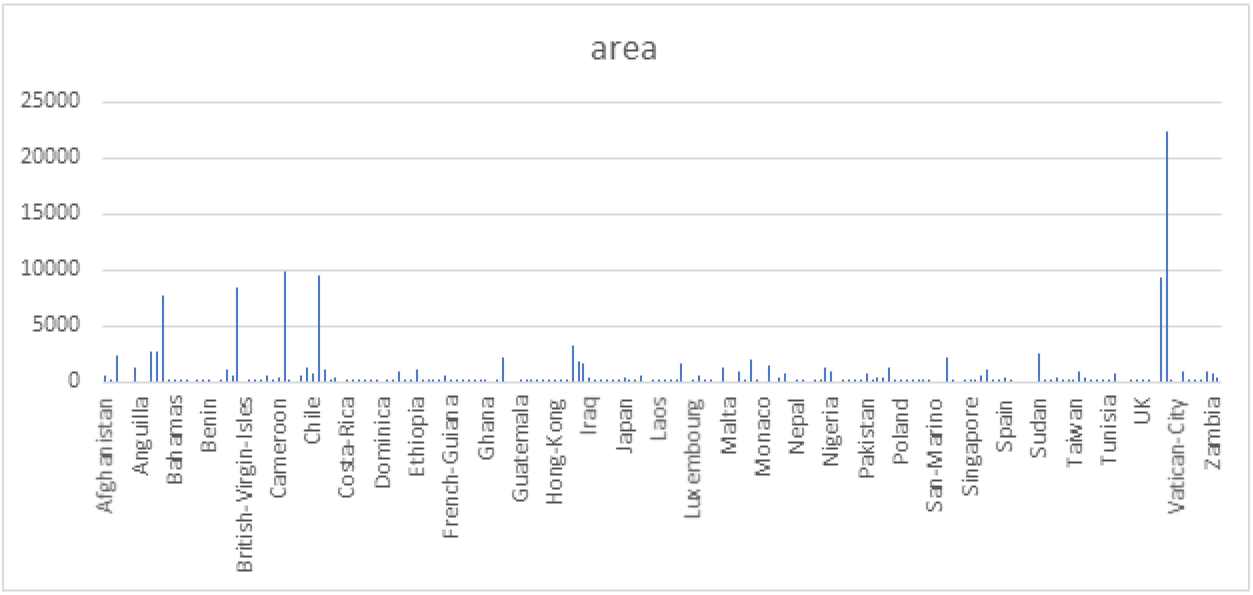
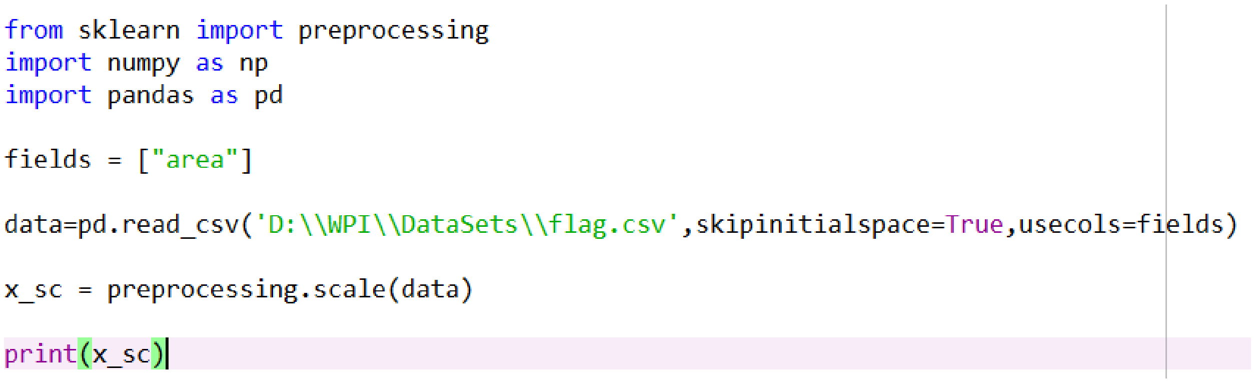


Fig 4.1 Original Graph

4.1 Standardization



```
from sklearn import preprocessing
import numpy as np
import pandas as pd

fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

x_sc = preprocessing.scale(data)

print(x_sc)
```

Fig 4.2 Code to Standardize area column using scale function

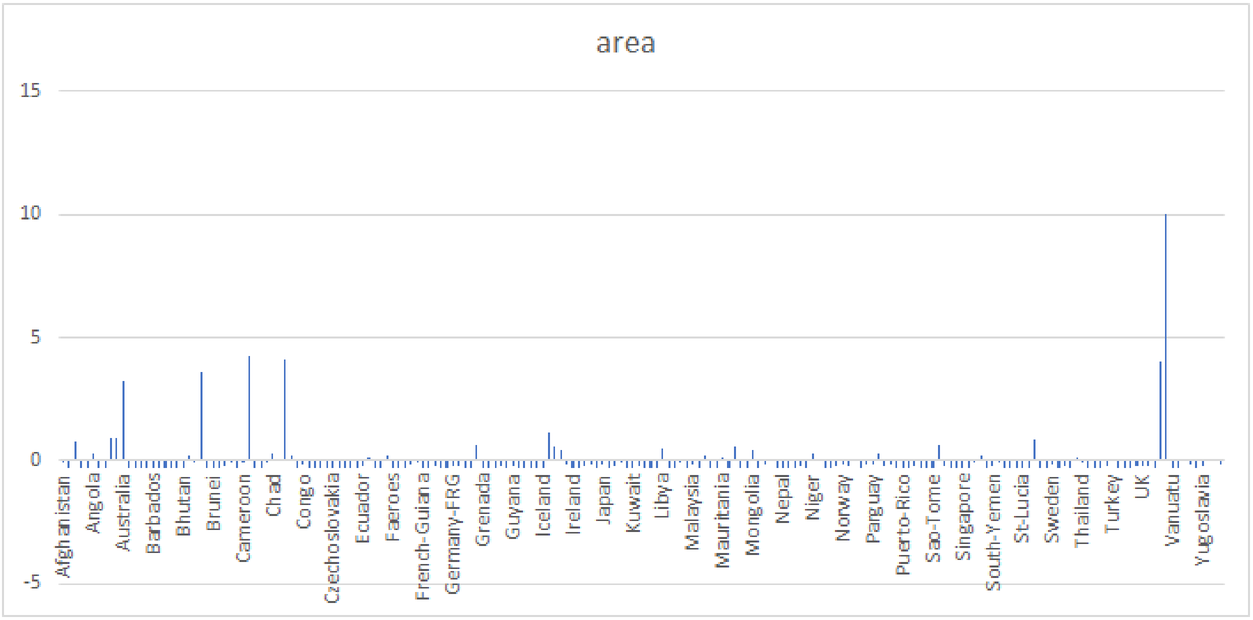


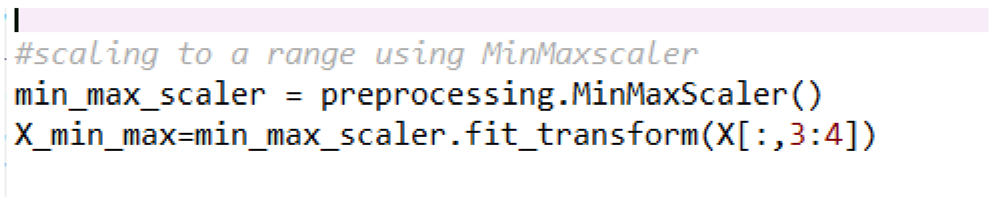
Fig 4.3 Graph after applying standardization

This function standardizes the data by transforming it to have zero mean and unit variance. The function used for this is as below:

$$x_{new} = x - \mu/\sigma$$

4.2 scaling to a range

- **Using MinMaxScaler**



```
#scaling to a range using MinMaxscaler
min_max_scaler = preprocessing.MinMaxScaler()
X_min_max=min_max_scaler.fit_transform(X[:,3:4])
```

Fig 4.4 Code for scaling to a range using MinMaxScaler

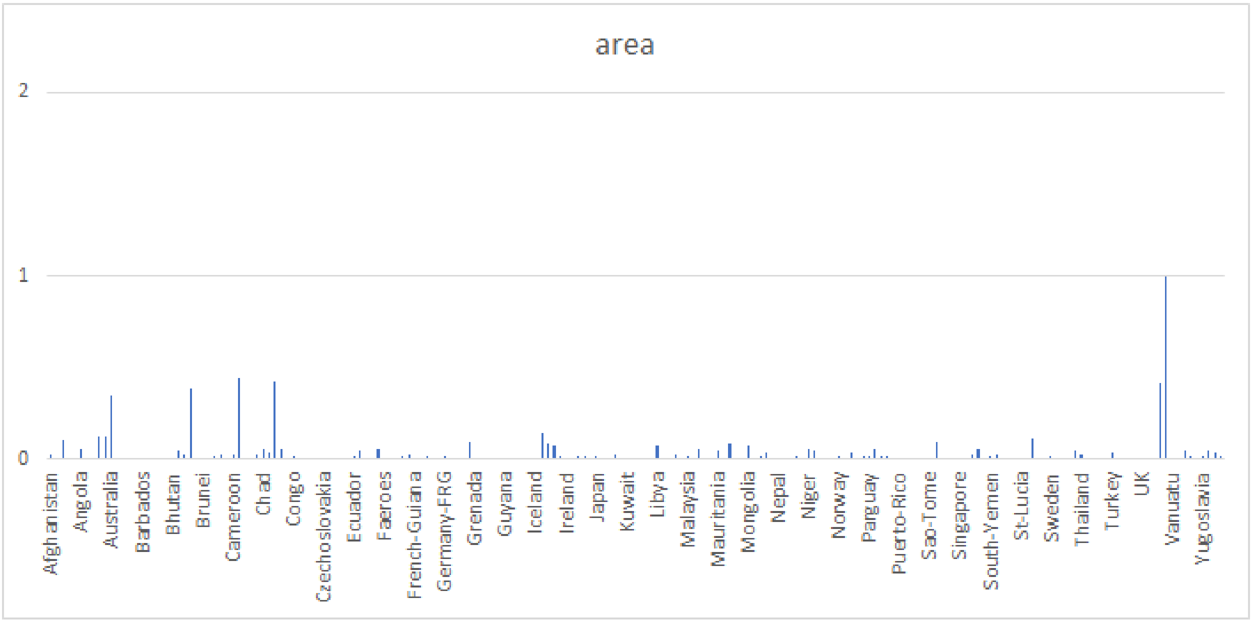
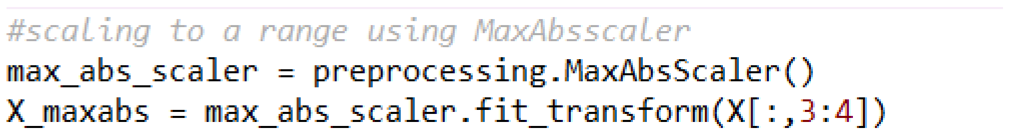


Fig 4.5 Graph for scaling to a range using MinMaxScalar

This function scales all the values by default within the range 0 to . This can be changed by specifying the range in the feature_range parameter

- **Using MaxAbsScaler**



```
#scaling to a range using MaxAbsscaler
max_abs_scaler = preprocessing.MaxAbsScaler()
X_maxabs = max_abs_scaler.fit_transform(X[:,3:4])
```

Fig 4.6 Code for scaling to a range using MaxAbsScaler

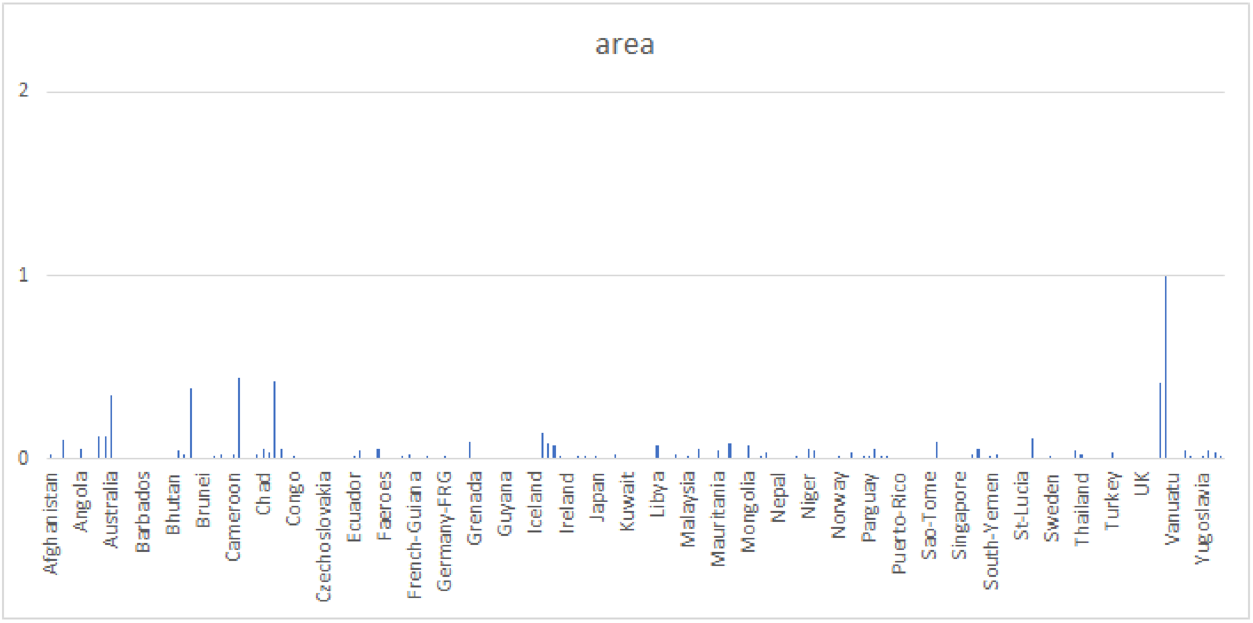
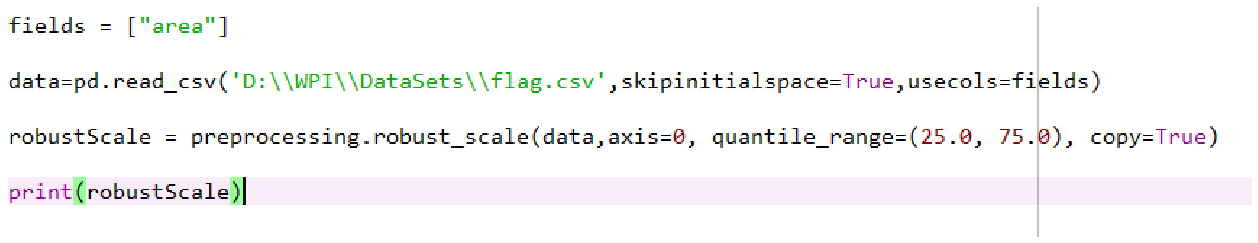


Fig 4.7 Graph of Scaling to a range using MaxAbsScaler

This function scales the feature based on the maximum absolute value in the particular feature being specified.

- **Using robust_scale**



```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

robustScale = preprocessing.robust_scale(data,axis=0, quantile_range=(25.0, 75.0), copy=True)

print(robustScale)
```

Fig 4.8 Code for scaling to a range using robust_scale

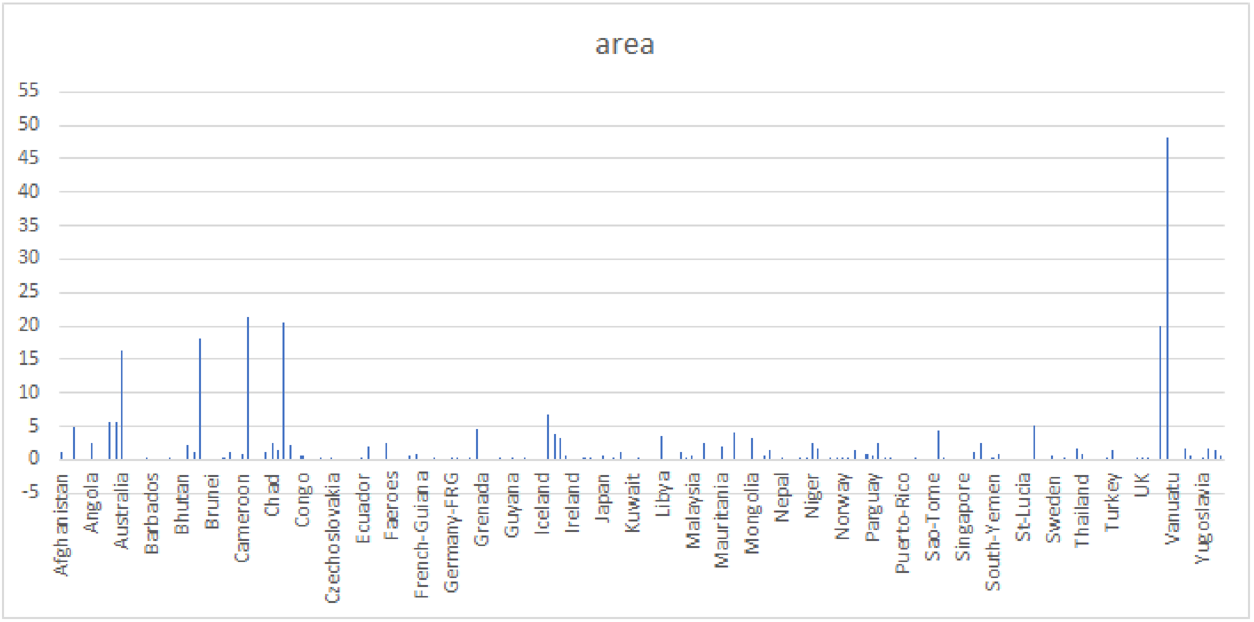
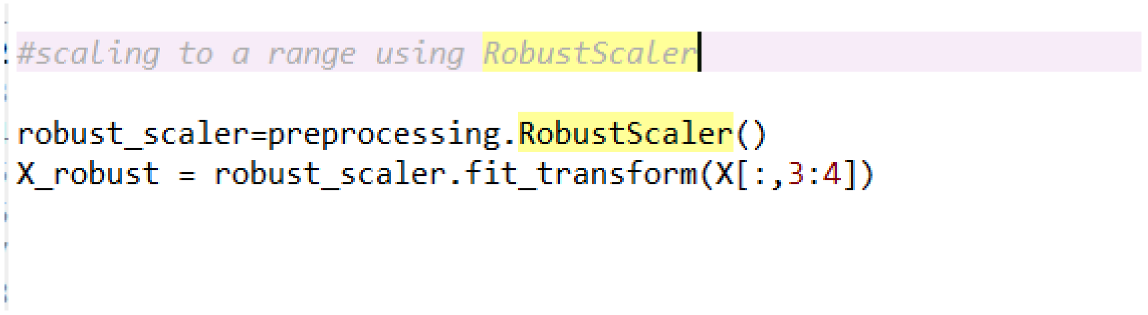


Fig 4.9 Graph of Scaling to a range using robust_scale

This scale standardizes data in any scale from center to the median according to the interquartile range.

- **Using RobustScaler**



```
#scaling to a range using RobustScaler

robust_scaler=preprocessing.RobustScaler()
X_robust = robust_scaler.fit_transform(X[:,3:4])
```

Fig 4.10 Code for scaling to a range using RobustScaler

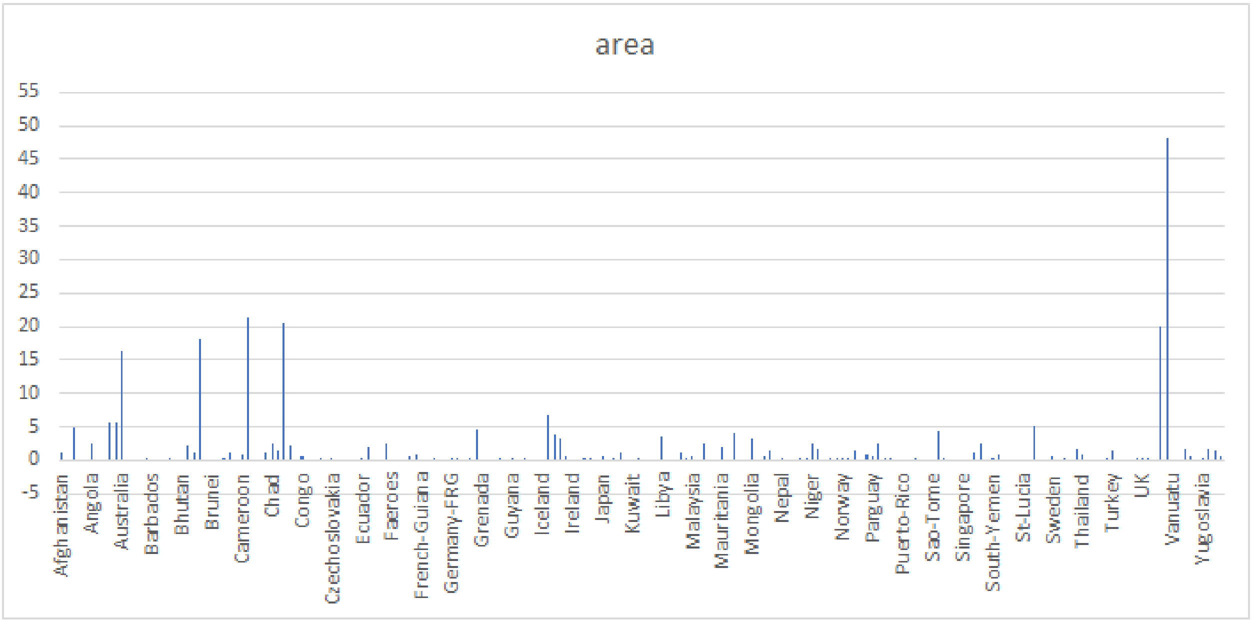
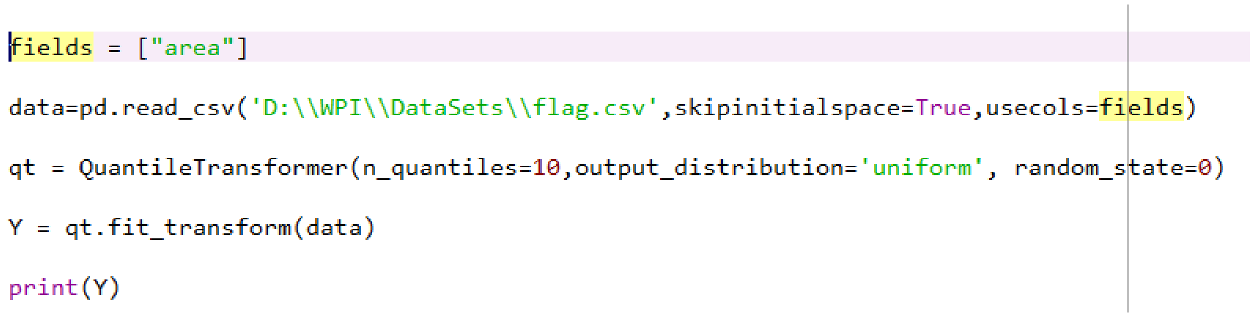


Fig 4.11 Graph of Scaling to a range using RobustScaler

This function scales the data based on quantile range (which by default is 1st and 3rd quantile) after removing the median

4.3 Mapping to a uniform distribution

- **Quantile Transformation**



```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

qt = QuantileTransformer(n_quantiles=10,output_distribution='uniform', random_state=0)

Y = qt.fit_transform(data)

print(Y)
```

Fig 4.12 Code for Uniform Distribution using Quantile Transformation

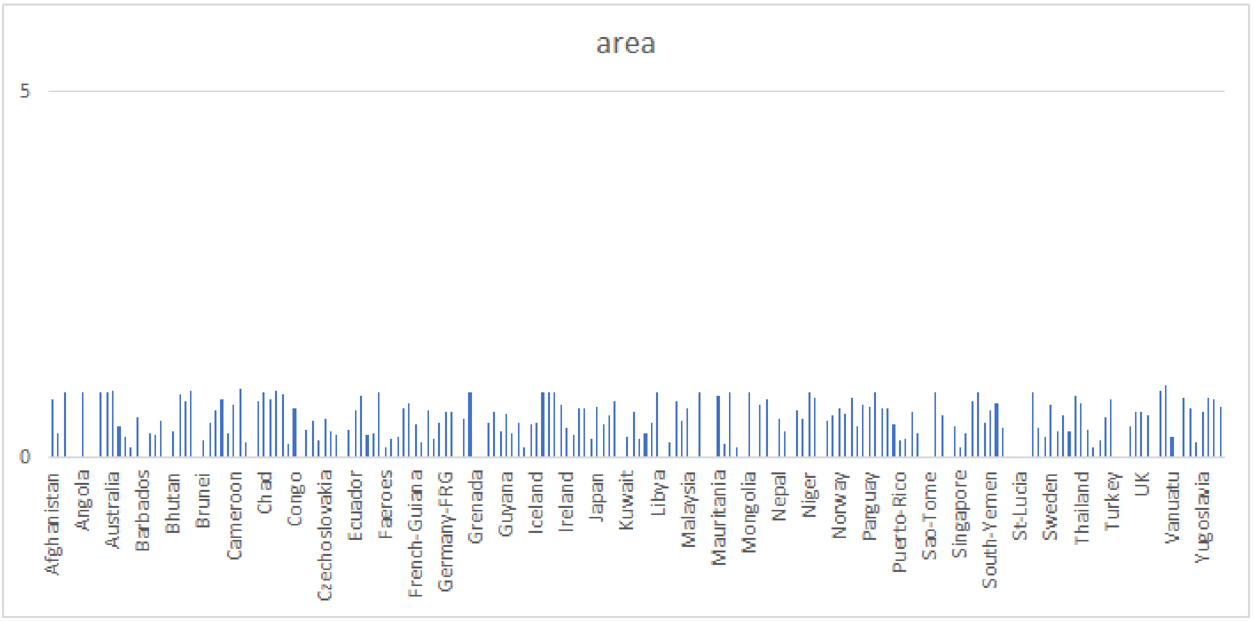


Fig 4.13 Graph of Uniform Distribution using Quantile Transformation

This function transforms the feature such that it follows either uniform or normal distribution. It reduces the outlier and therefore is a robust preprocessing technique.

- **Quantile_transform**

```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

qt = quantile_transform(data,n_quantiles=10,output_distribution='uniform', random_state=0)

print(qt)
```

Fig 4.14 Code for Uniform Distribution using quantile_transform

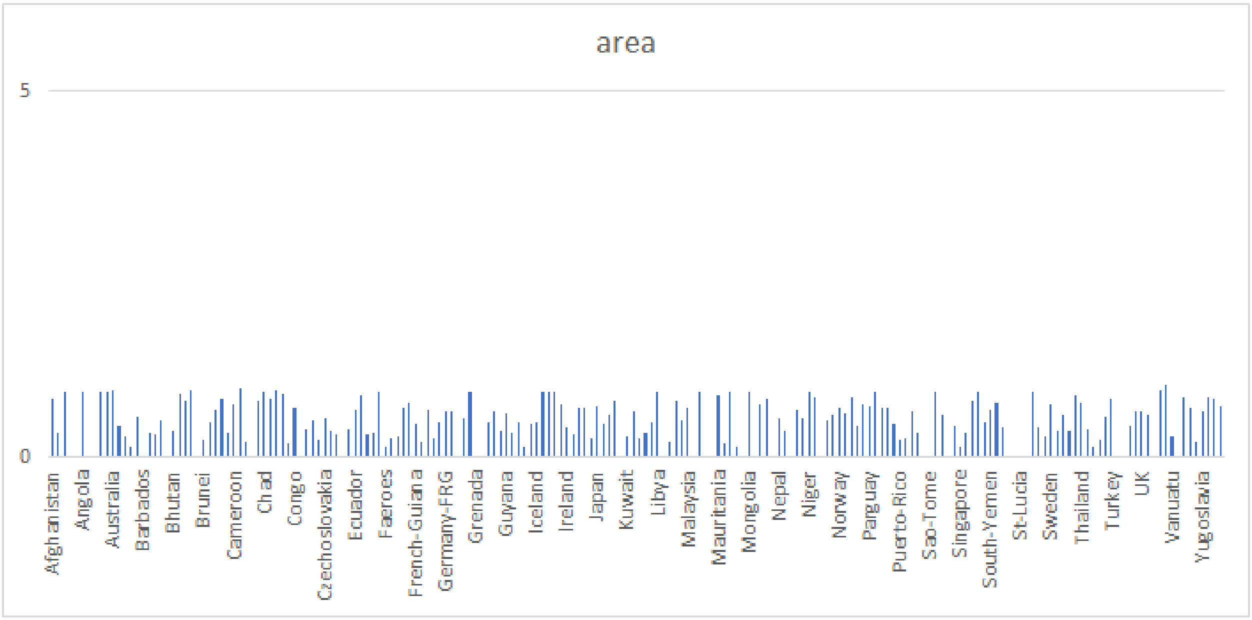
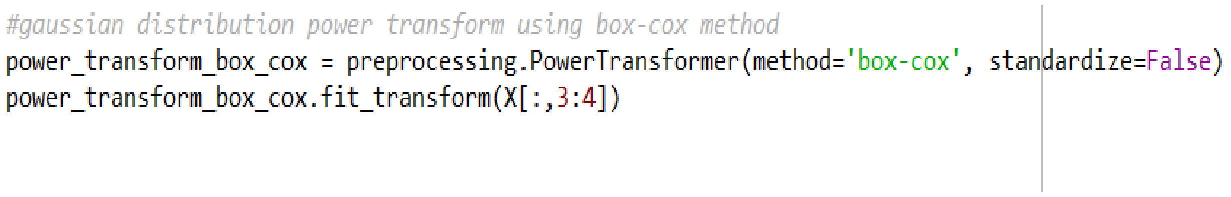


Fig 4.15 Graph of Uniform Distribution using quantile_transform

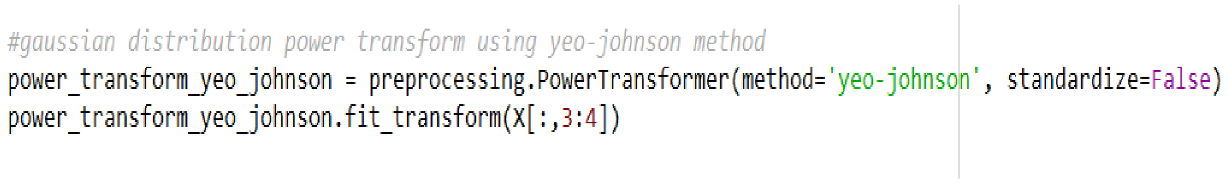
This function converts the column area to uniform distribution such that all the values lie between 0 and 1

4.4 Mapping to a Gaussian distribution

- **Using PowerTransform**



```
#gaussian distribution power transform using box-cox method
power_transform_box_cox = preprocessing.PowerTransformer(method='box-cox', standardize=False)
power_transform_box_cox.fit_transform(X[:,3:4])
```



```
#gaussian distribution power transform using yeo-johnson method
power_transform_yeo_johnson = preprocessing.PowerTransformer(method='yeo-johnson', standardize=False)
power_transform_yeo_johnson.fit_transform(X[:,3:4])
```

Fig 4.16 Code of Gaussian Distribution using power_transform using two methods

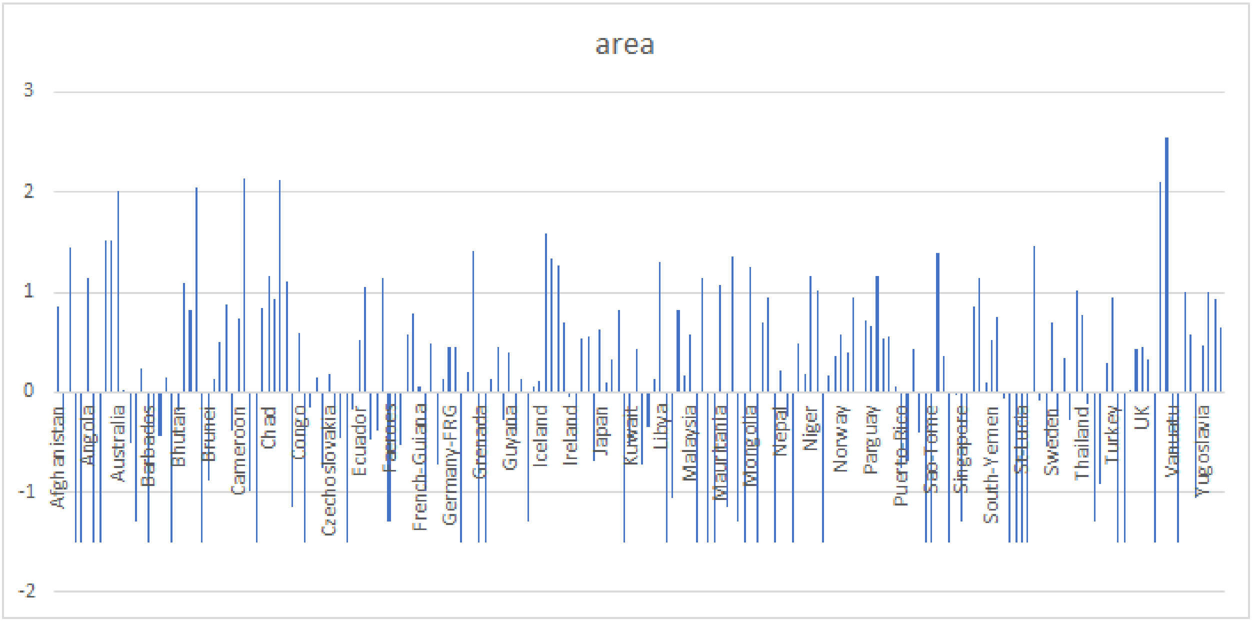
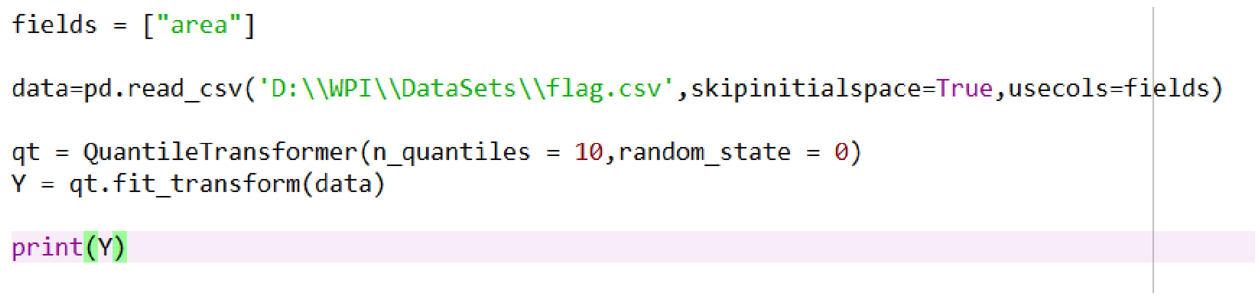


Fig 4.17 Graph of Gaussian Distribution using power_transform

The power transform is used to transform the data to be more gaussian to model data that have non_constant variance.

- **Using QuantileTransformer**



```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

qt = QuantileTransformer(n_quantiles = 10,random_state = 0)
Y = qt.fit_transform(data)

print(Y)
```

Fig 4.18 Code for Gaussian Distribution using QuantileTransformer

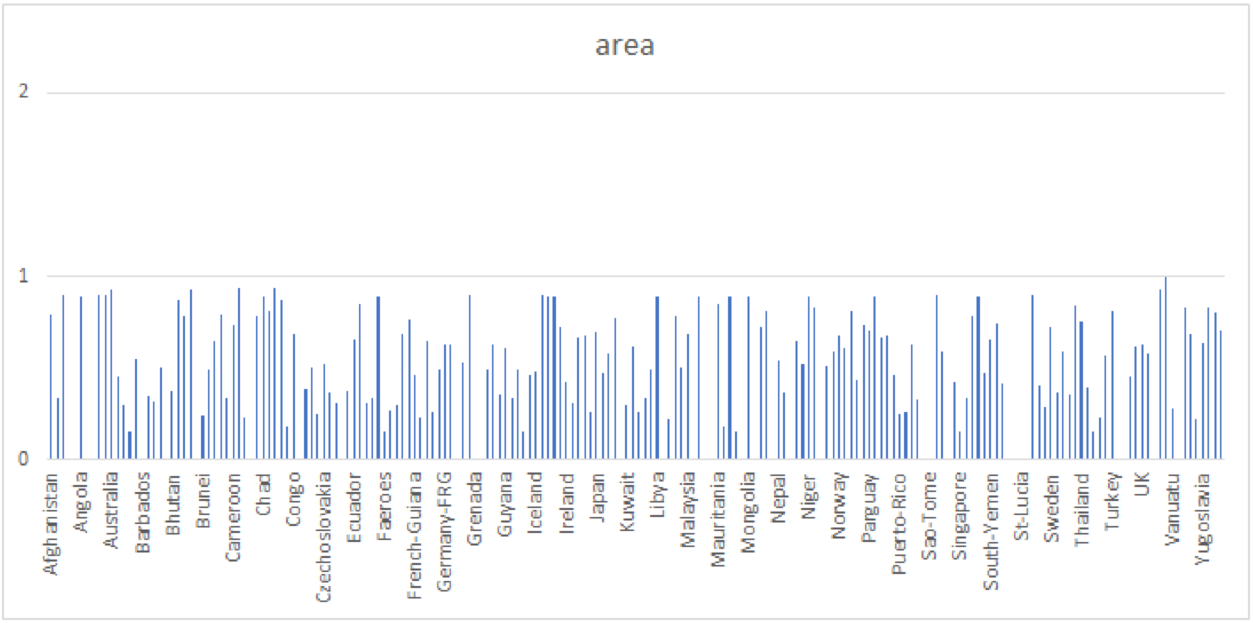


Fig 4.19 Graph of Gaussian Distribution using QuantileTransformer

4.5 Normalization

```
fields = ["area"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

normalizer = preprocessing.Normalizer().fit(data)

Y = normalizer.transform(data)

print(Y)
```

Fig 420. Code of Normalization

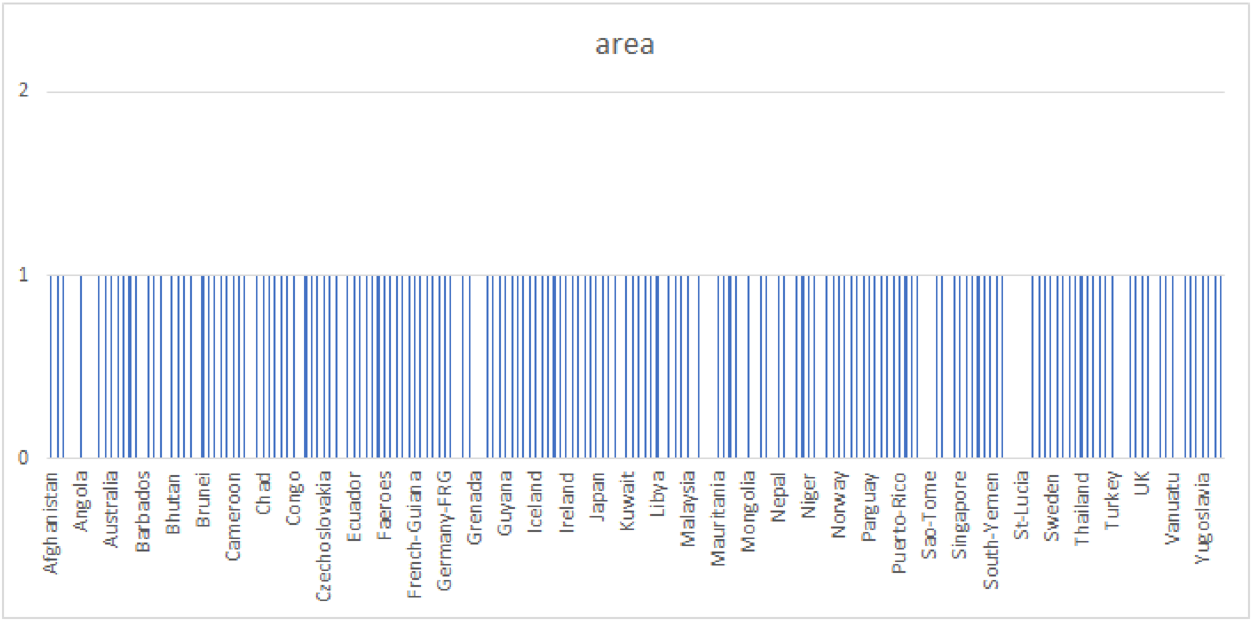


Fig 4.21 Graph of Normalization

This function uses normalization on column area (#4) and scales individual samples to unit.

5. Discretization

- **Original Graph**

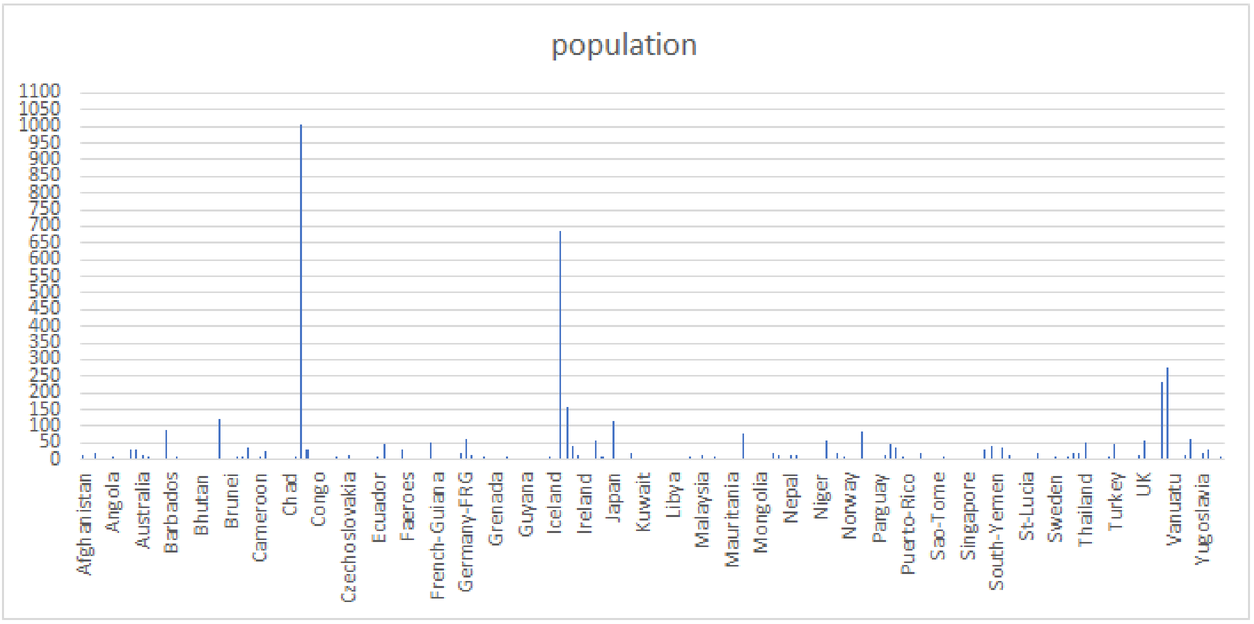
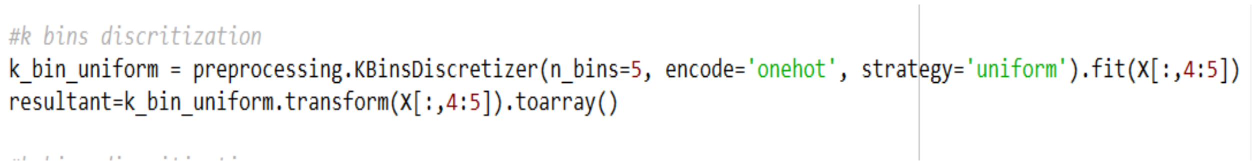


Fig 5.1 Original Graph

1. **K-Bins discretization**
   - **Encoding :** onehot , onehot-dense ,ordinal
   - **Strategy :** uniform , quantile , kmeans

❑ **n_bins=5 , encode= onehot , strategy= uniform**



```
#k bins discritization
k_bin_uniform = preprocessing.KBinsDiscretizer(n_bins=5, encode='onehot', strategy='uniform').fit(X[:,4:5])
resultant=k_bin_uniform.transform(X[:,4:5]).toarray()
```

Fig 5.2 Code to discretize into 5 bins using one hot encoding

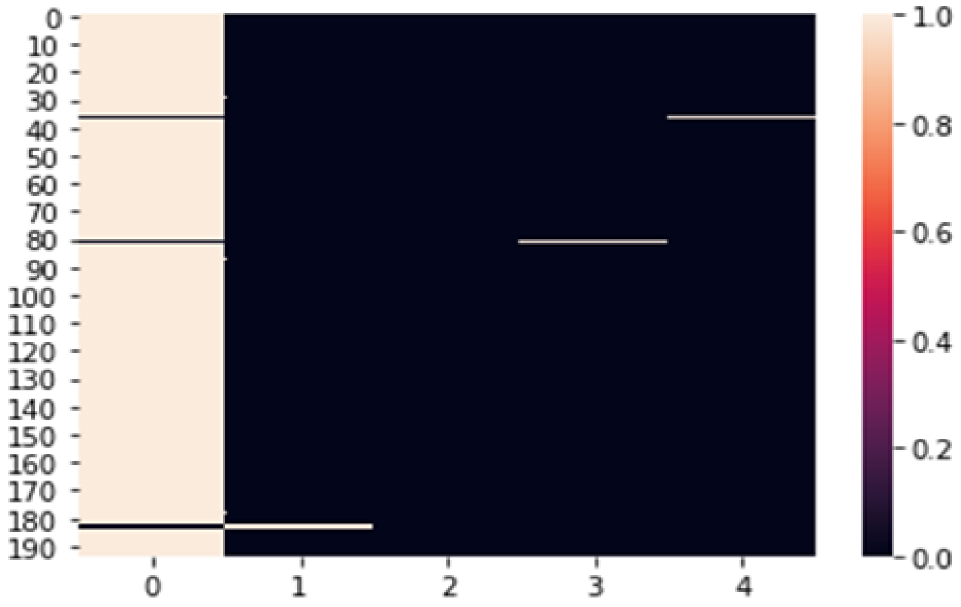
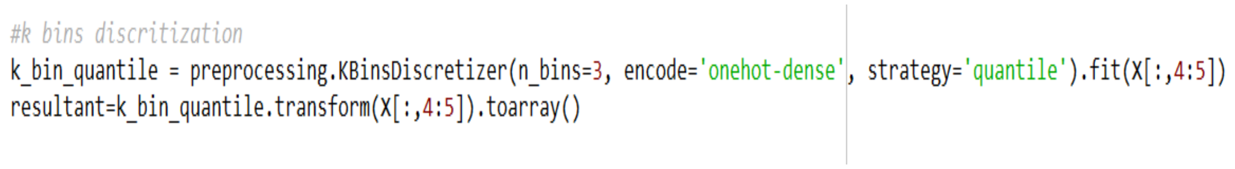


Fig 5.3 Plot after discretization

- ❑ **n_bins=3, encode= onehot-dense , strategy=quantile**



```
#k bins discritization
k_bin_quantile = preprocessing.KBinsDiscretizer(n_bins=3, encode='onehot-dense', strategy='quantile').fit(X[:,4:5])
resultant=k_bin_quantile.transform(X[:,4:5]).toarray()
```

Fig 5.4 Code to discretize into 3 bins using onehot-dense encoding

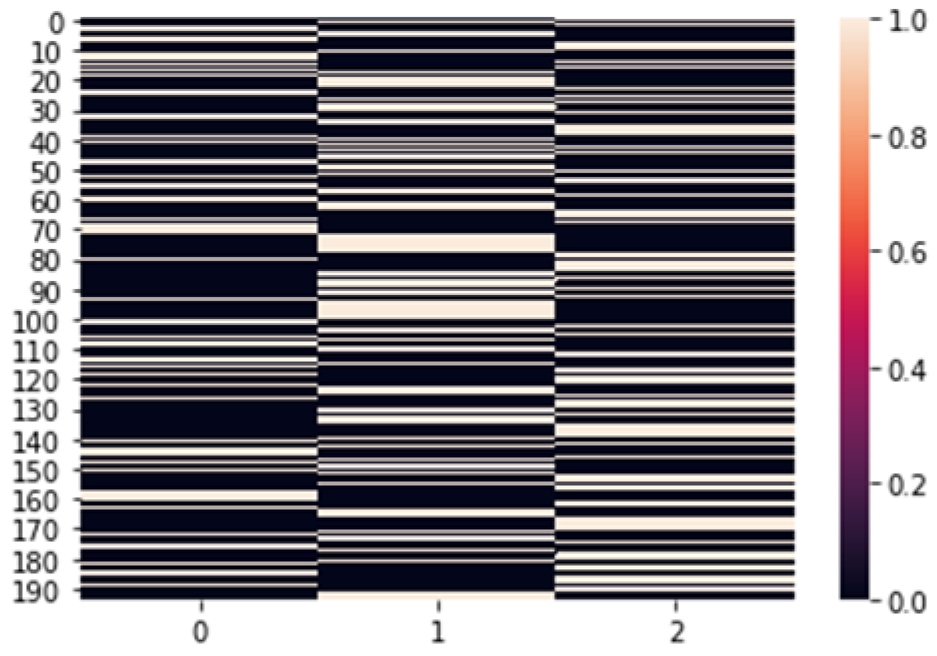
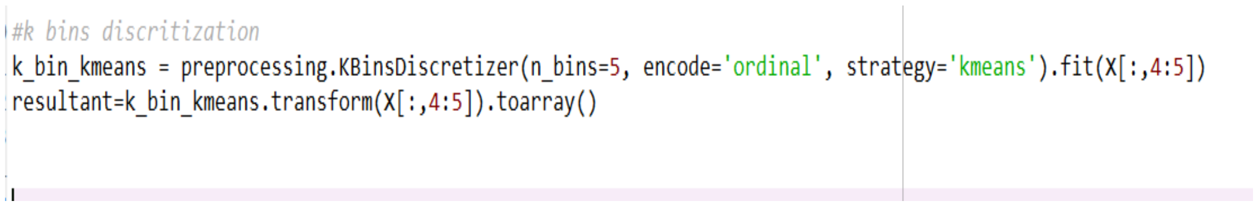


Fig 5.5 Plot after discretization

❏ **n_bins=5 , encode=ordinal , strategy=kmeans**



```
#k bins discritization
k_bin_kmeans = preprocessing.KBinsDiscretizer(n_bins=5, encode='ordinal', strategy='kmeans').fit(X[:,4:5])
resultant=k_bin_kmeans.transform(X[:,4:5]).toarray()
```

Fig 5.6 Code to discretize into 5bins using ordinal encoding

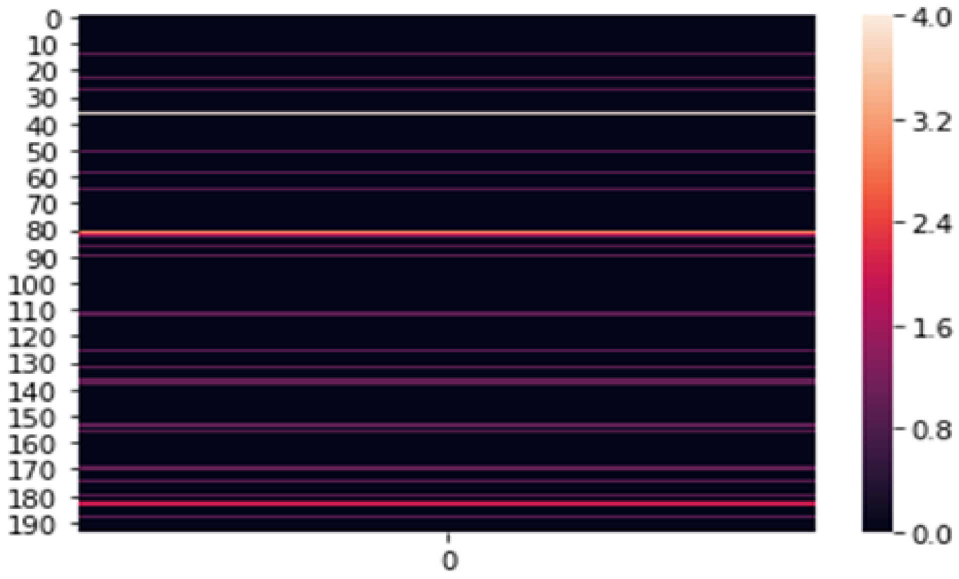


Fig 5.7 Plot after discretization

2. Feature Binarization

Feature Binarization, converts the data to binary i.e either to 0 or based on the threshold passed as parameter. Here, we have experimented with the below thresholds

- **Threshold=0.0**

```
fields = ["population"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

binarize = Binarizer(threshold=0.0).fit(data)
Y = binarize.transform(data)

print(Y)
```

Fig 5.8 Code of Feature Binarization with Threshold = 0.0

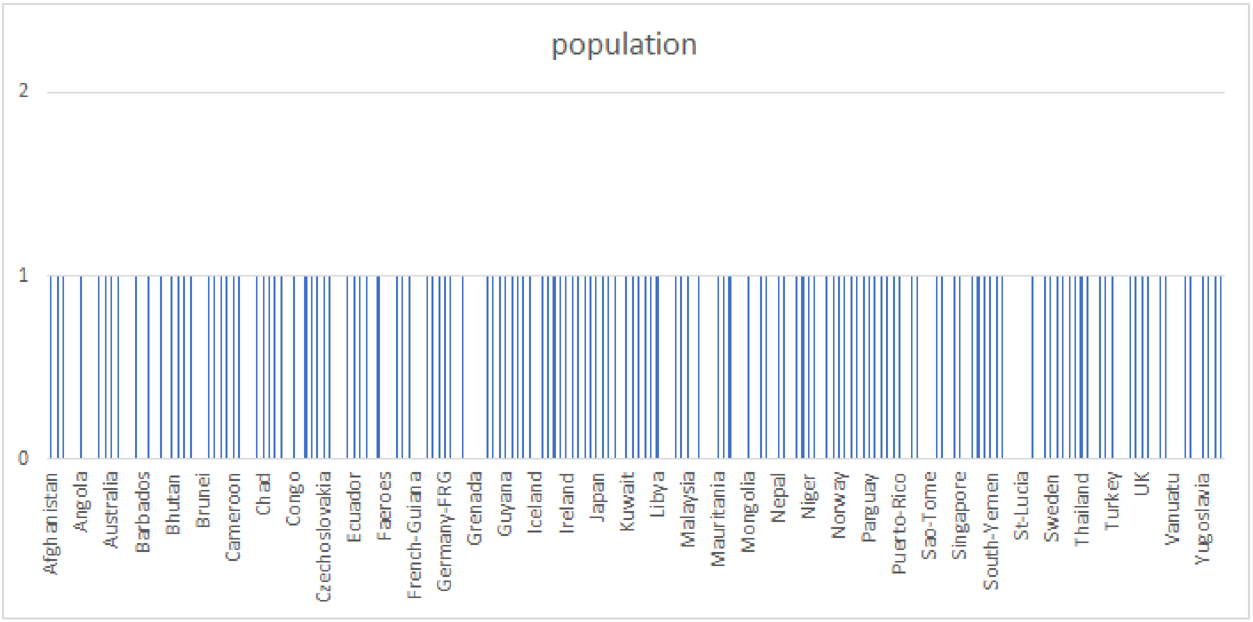
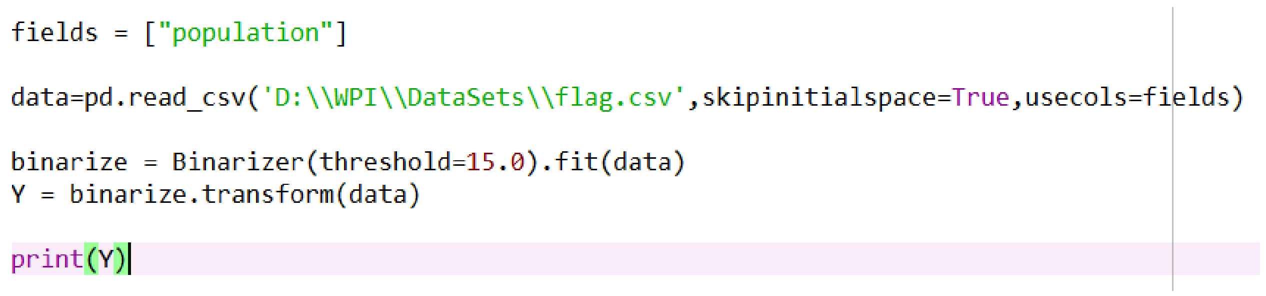


Fig 5.9 Graph of Feature Binarization with Threshold = 0.0

- **Threshold = 15.0**



```
fields = ["population"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

binarize = Binarizer(threshold=15.0).fit(data)
Y = binarize.transform(data)

print(Y)
```

Fig 5.10 Code of Feature Binarization with Threshold = 15.0

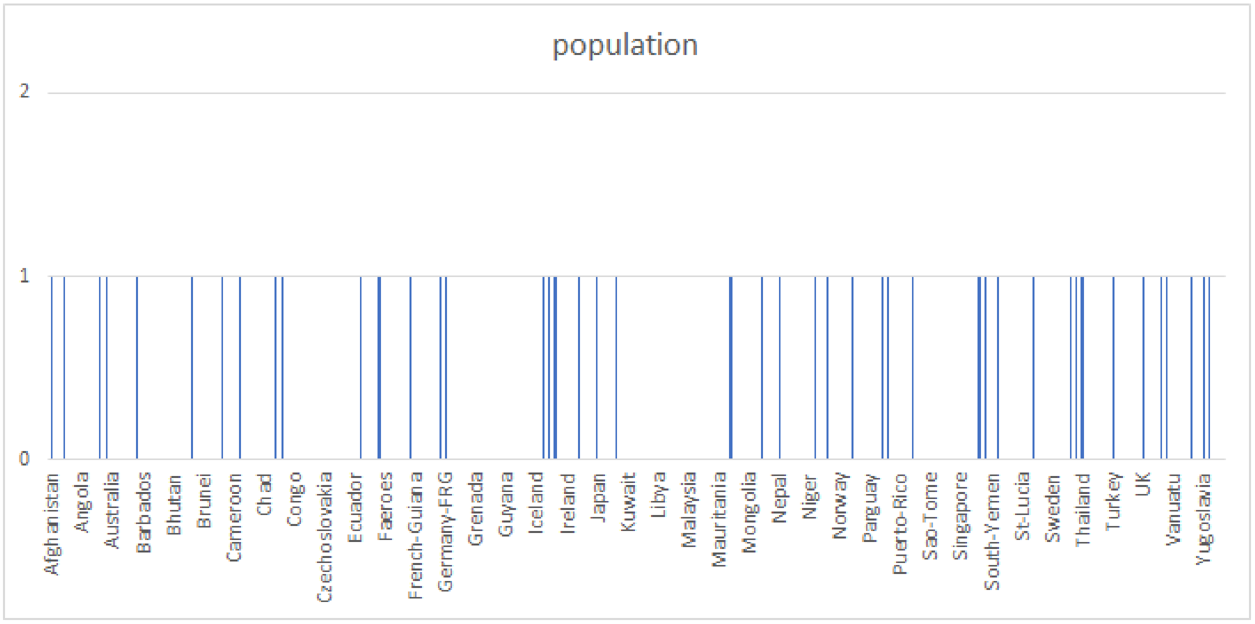
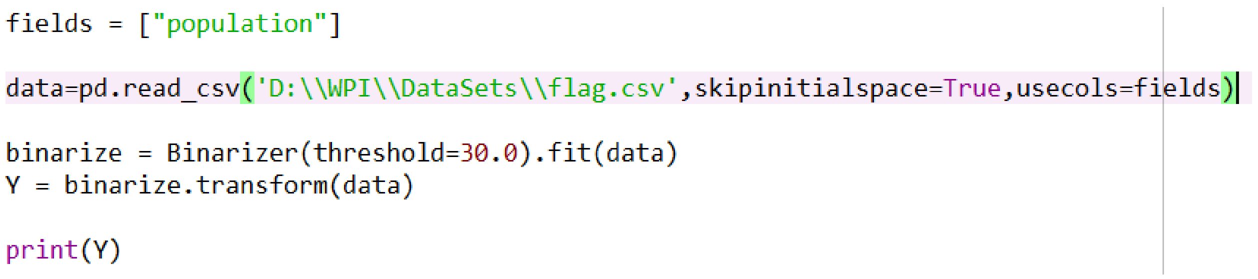


Fig 5.11 Graph of Feature Binarization with Threshold = 15.0

- **Threshold = 30.0**



```
fields = ["population"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

binarize = Binarizer(threshold=30.0).fit(data)
Y = binarize.transform(data)

print(Y)
```

Fig 5.12 Code of Feature Binarization with Threshold = 30.0

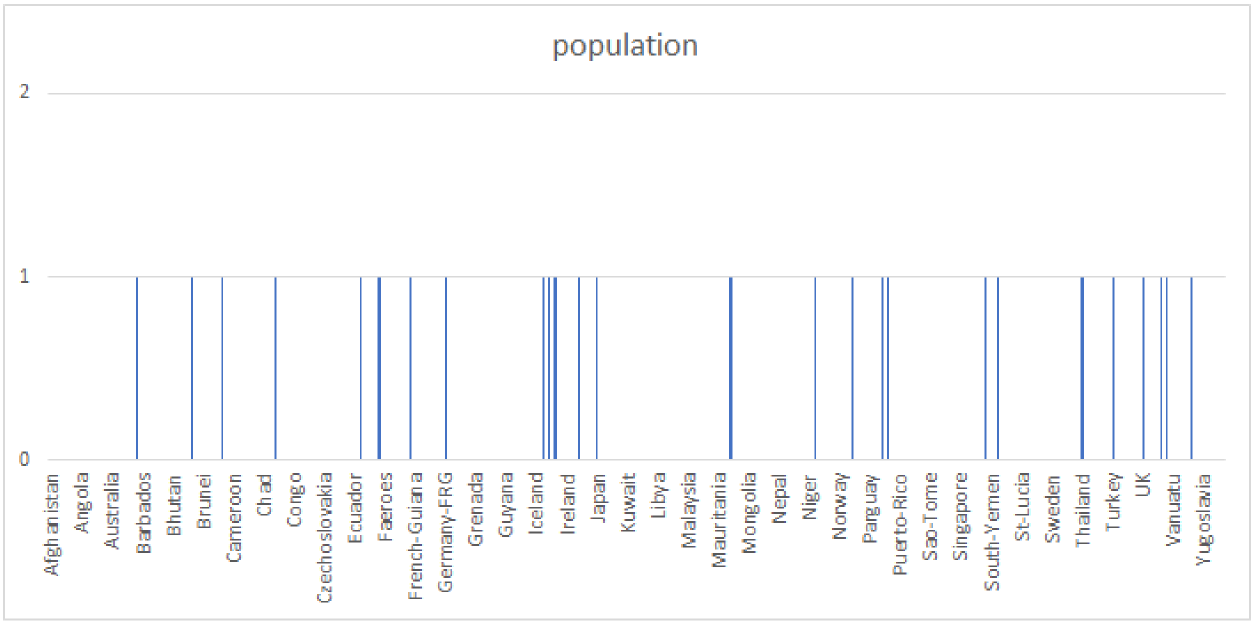
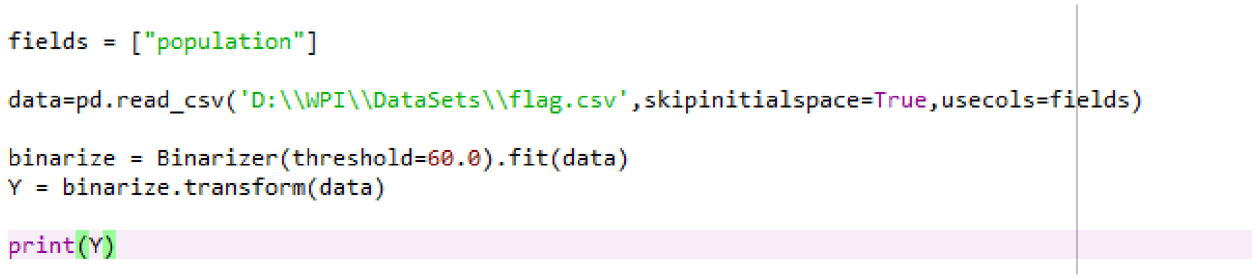


Fig 5.13 Graph of Feature Binarization with Threshold = 30.0

- **Threshold = 60.0**



```
fields = ["population"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

binarize = Binarizer(threshold=60.0).fit(data)
Y = binarize.transform(data)

print(Y)
```

Fig 5.14 Code of Feature Binarization with Threshold = 60.0

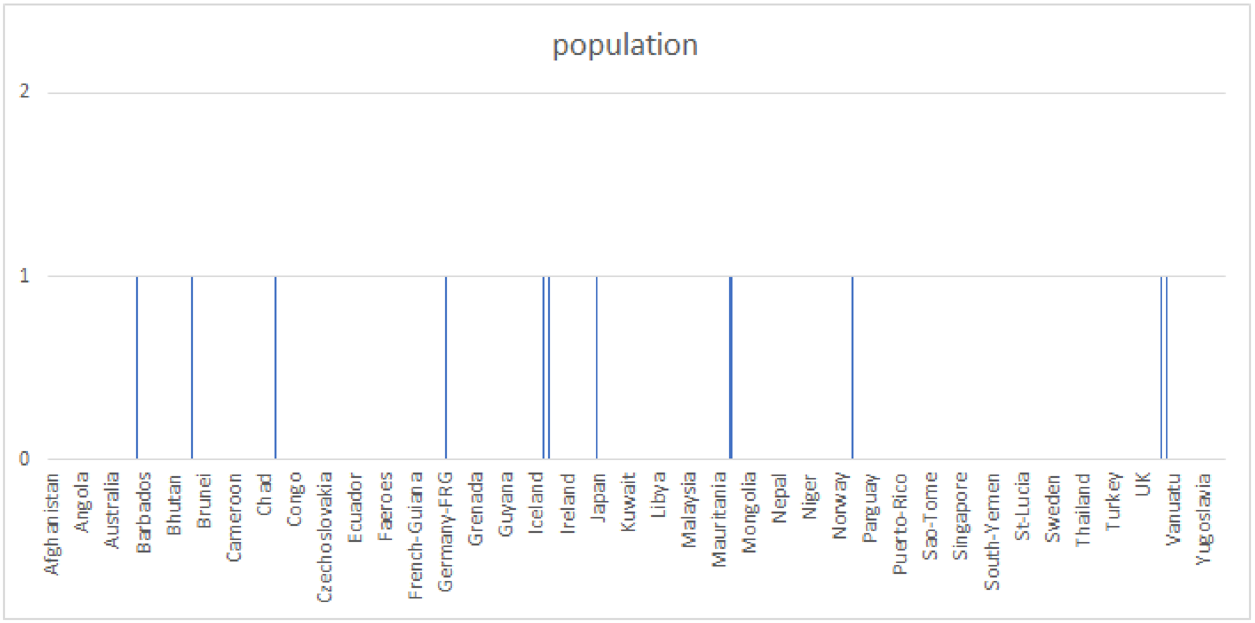


Fig 5.15 Graph of Feature Binarization with Threshold = 60.0

6. Custom Transformation

- **Original Graph**

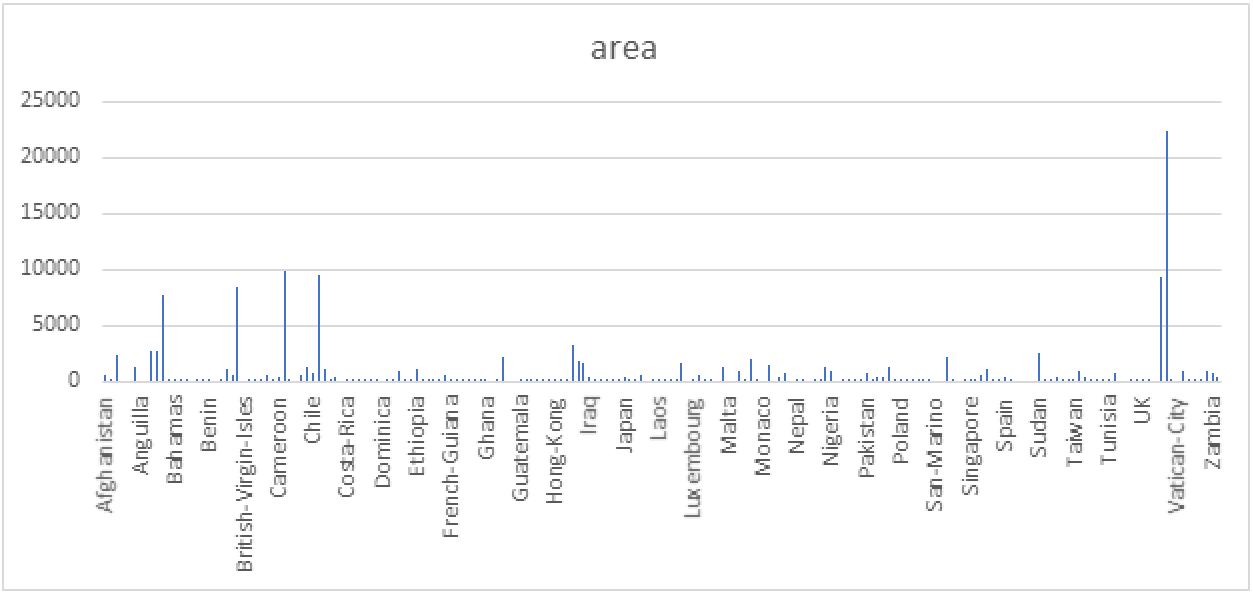


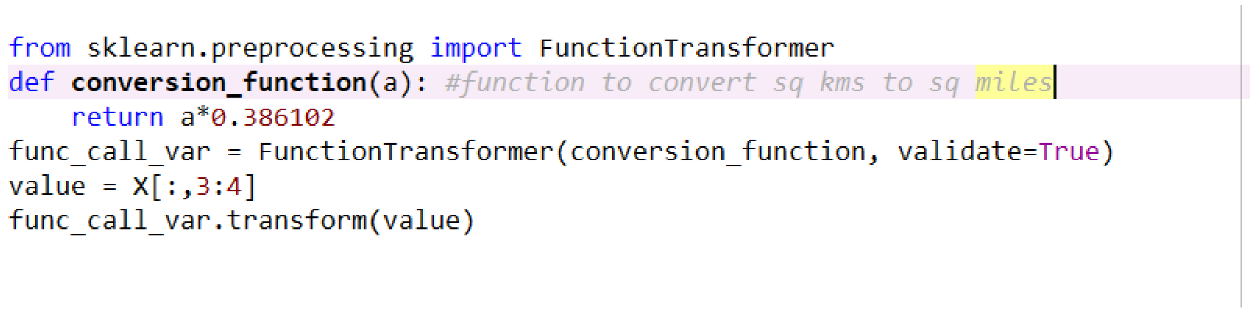
Fig 6.1 Graph of Original data

- **Function Transformer**

  This function transforms the data by forwarding the attribute to a user defined function and returns the results.

  Here the function is invoked with two parameters namely, the user defined function name and validate which validates every input before sending it to the user defines function when set to true.

  The user defined function multiplies each input by a factor(0.386102) to convert to sq miles and returns the value.



```
from sklearn.preprocessing import FunctionTransformer
def conversion_function(a): #function to convert sq kms to sq miles
    return a*0.386102
func_call_var = FunctionTransformer(conversion_function, validate=True)
value = X[:,3:4]
func_call_var.transform(value)
```

Fig 6.2 Code of Function Transformer

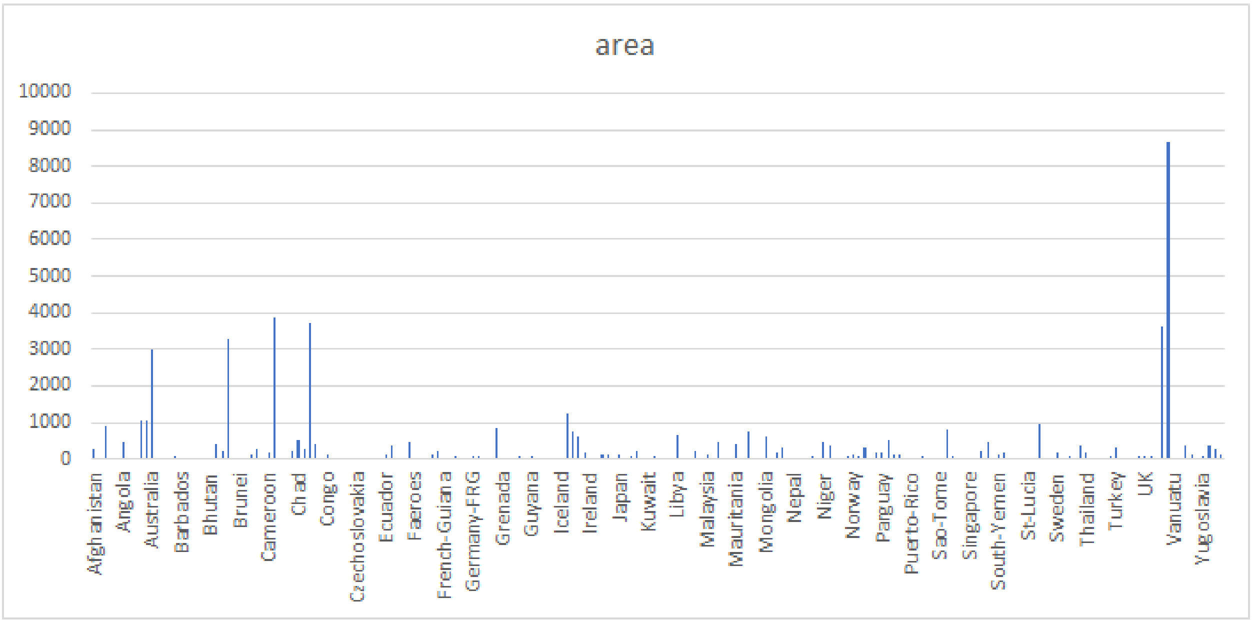
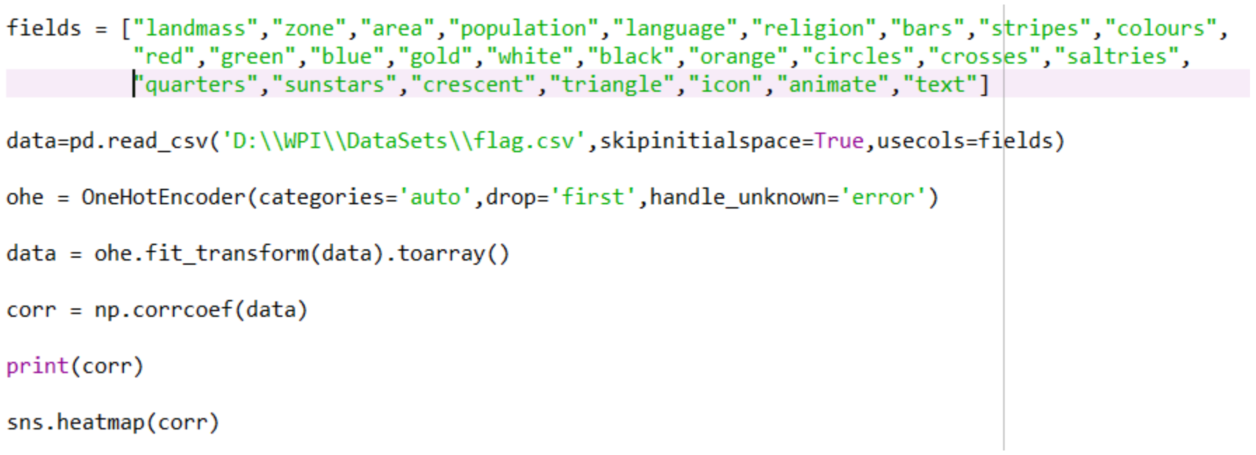


Fig 6.3 Graph after applying Function Transformer

Problem III Data Preprocessing : Dimensionality Reduction

1. **Correlation and Covariance Analysis**
   1) **Correlation Matrix**
      - **Correlation Matrix of the dataset**



```
fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

ohe = OneHotEncoder(categories='auto',drop='first',handle_unknown='error')

data = ohe.fit_transform(data).toarray()

corr = np.corrcoef(data)

print(corr)

sns.heatmap(corr)
```

Fig 1.1 Code of Correlation using numpy

- **Visualization of the Matrix**

  The visualization of this correlation is done using heatmap, the code is attached in Fig 1.1

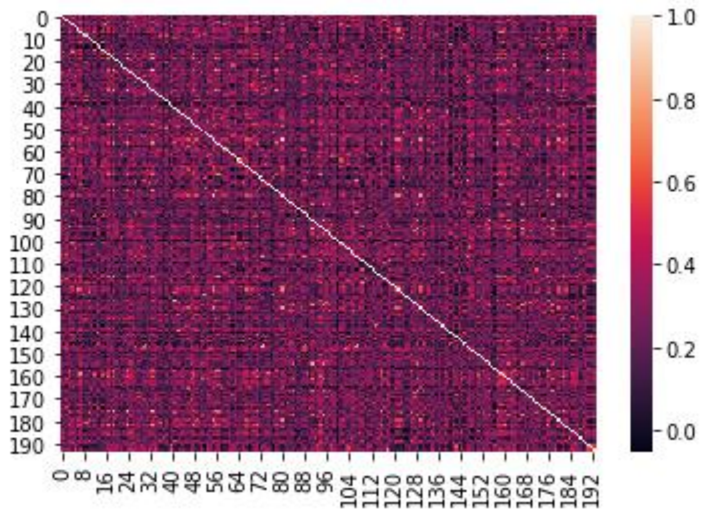
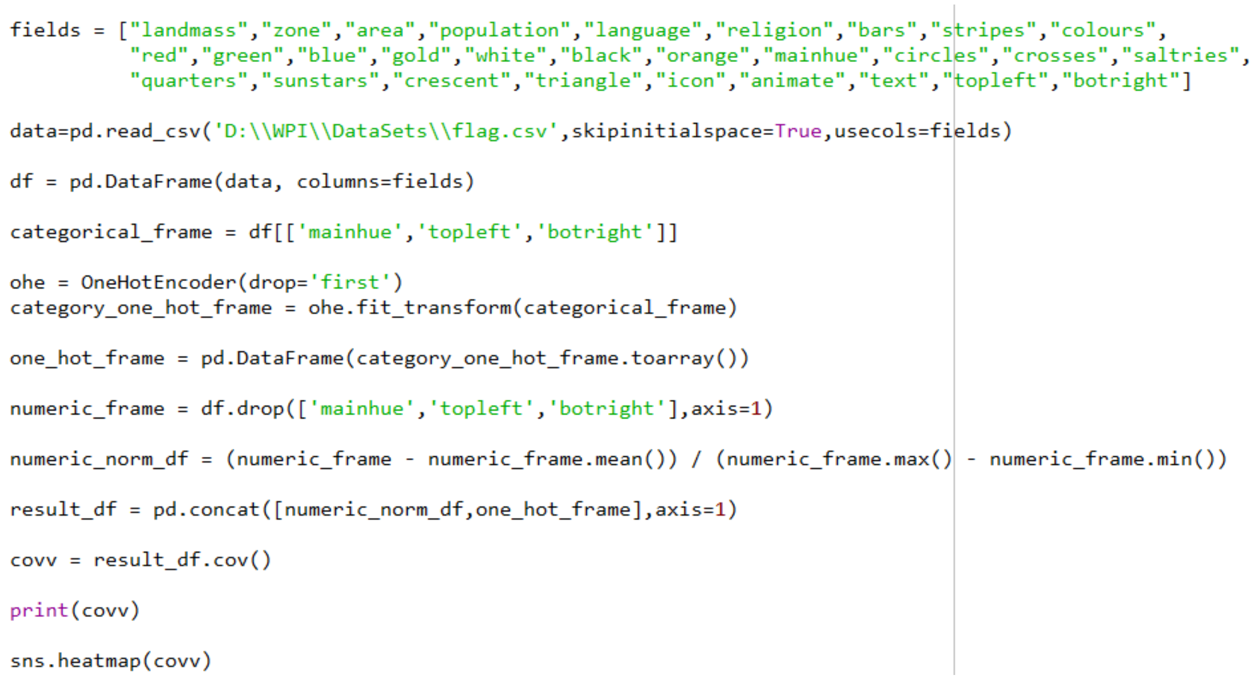


Fig 1.2 Visualization of Correlation Matrix

Note: The correlation matrix is attached in a separate xls file named Correlation_matrix.xlsx

2) Covariance Matrix

- **Covariance Matrix of the dataset**



```
fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

numeric_norm_df = (numeric_frame - numeric_frame.mean()) / (numeric_frame.max() - numeric_frame.min())

result_df = pd.concat([numeric_norm_df,one_hot_frame],axis=1)

covv = result_df.cov()

print(covv)

sns.heatmap(covv)
```

Fig 1.3 Code of Covariance using panda's dataframe

- **Visualization of the Matrix**

  The visualization of this correlation is done using heatmap, the code is attached in Fig 1.1

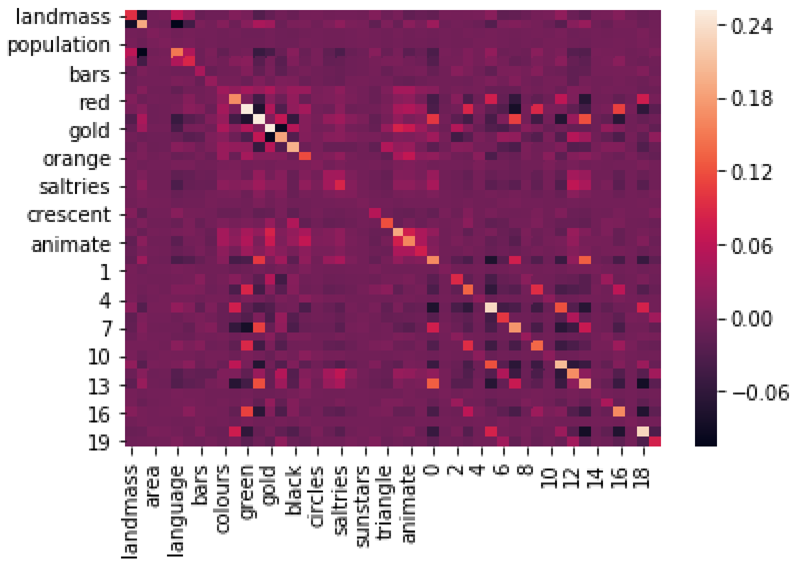


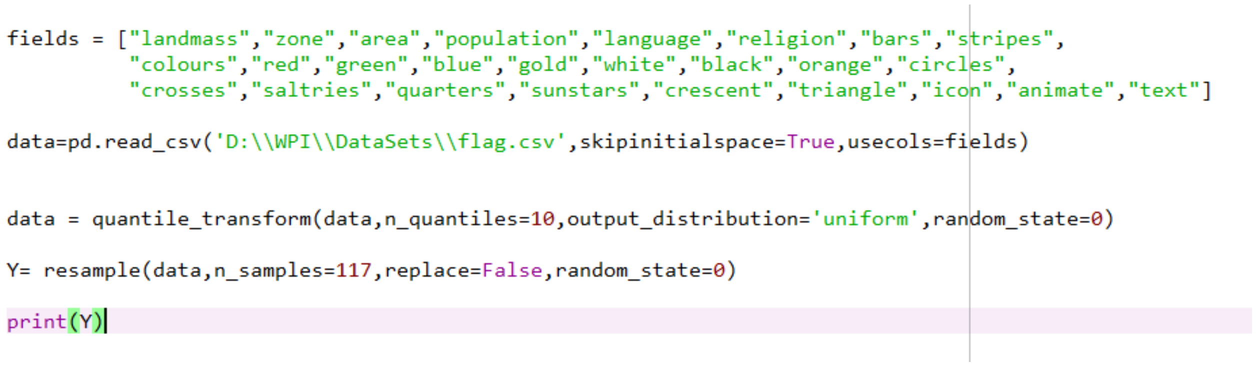
Fig 1.4 Visualization of Covariance Matrix

Note: The covariance matrix is attached in a separate xls file named covariance_matrix.xlsx

2. Data Sampling

Here, since we have to reduce the data instances just to keep 60% of the entire dataset, we have considered n_samples = 117

- **Random Sampling without replacement using uniform distribution**

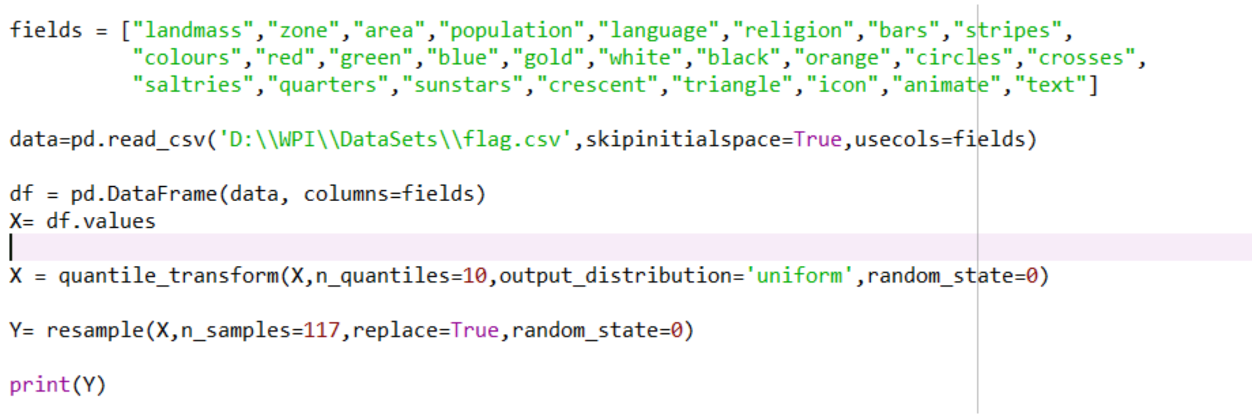


```
fields = ["landmass","zone","area","population","language","religion","bars","stripes",
          "colours","red","green","blue","gold","white","black","orange","circles",
          "crosses","saltries","quarters","sunstars","crescent","triangle","icon","animate","text"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)


data = quantile_transform(data,n_quantiles=10,output_distribution='uniform',random_state=0)

Y= resample(data,n_samples=117,replace=False,random_state=0)

print(Y)
```

Fig 2.1 Code for Random Sampling without replacement using uniform distribution

To apply random Sampling, we first have to quantile_transform the dataset and then apply resample without replacement.
The python libraries used for this sampling technique are quantile_transform, resample

- **Random Sampling with replacement using uniform distribution**

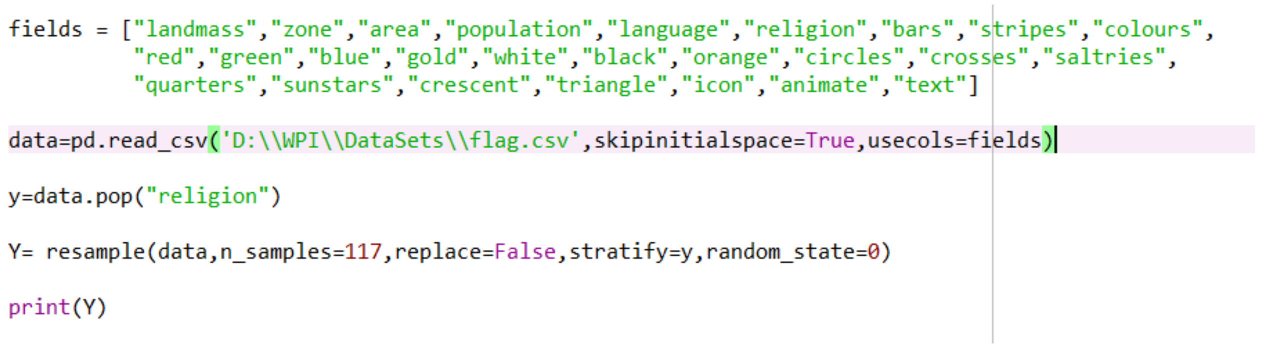


```
fields = ["landmass","zone","area","population","language","religion","bars","stripes",
          "colours","red","green","blue","gold","white","black","orange","circles","crosses",
          "saltries","quarters","sunstars","crescent","triangle","icon","animate","text"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)
X= df.values

X = quantile_transform(X,n_quantiles=10,output_distribution='uniform',random_state=0)

Y= resample(X,n_samples=117,replace=True,random_state=0)

print(Y)
```

Fig 2.2 Code for Random Sampling with replacement using uniform distribution

Random Sampling with replacement is the same as that without replacement except for the fact that in random sampling without replacement each sample has only one chance to be selected, whereas in random sampling without replacement there is an equal chance for every sample to be selected even after it being already selected.

- **Stratified Random Sampling without replacement using religion as target attribute**



```
fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

y=data.pop("religion")

Y= resample(data,n_samples=117,replace=False,stratify=y,random_state=0)

print(Y)
```

Fig 2.3 Stratified Random Sampling without replacement

In stratified random sampling the data is split into smaller samples also known as stratas basted on the target feature in such a way that each category gets equal samples.

3. **Feature Selection**
   - **Manual Feature Selection**

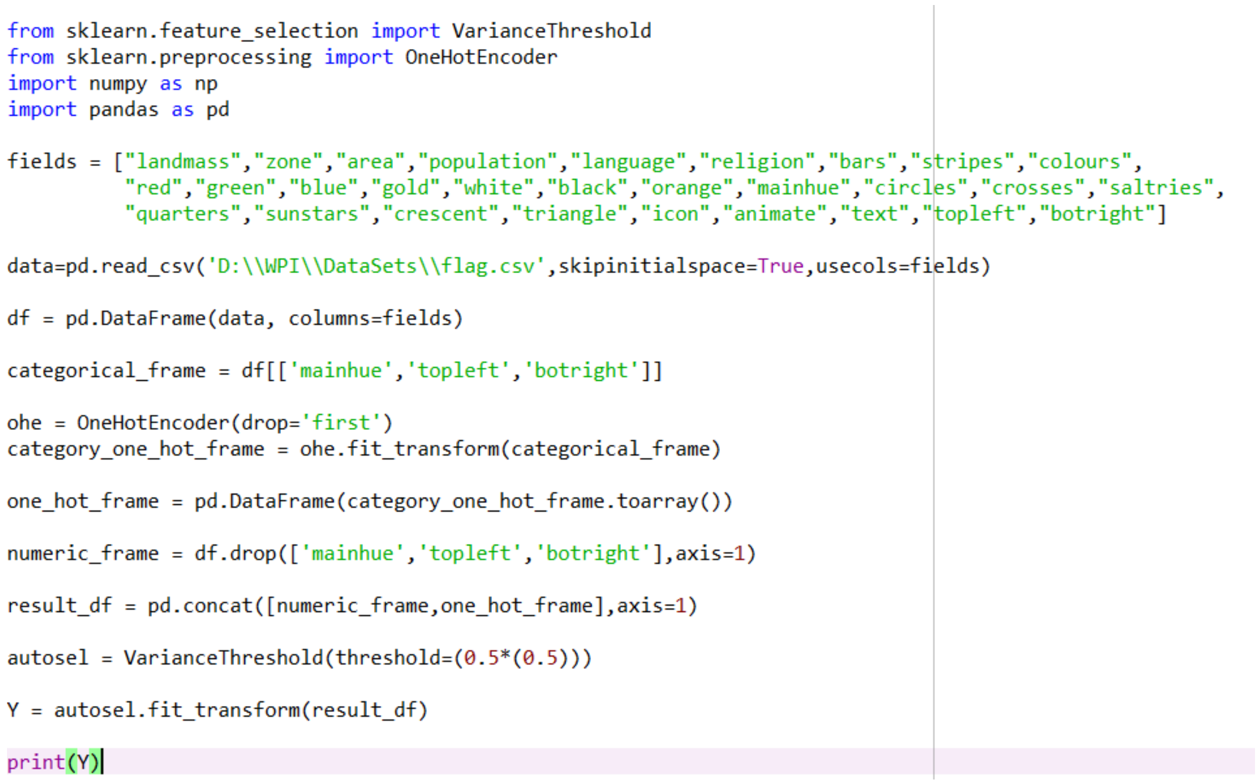
     Inorder to remove two features we can remove attributes that either have high positive correlation or high negative correlation.

     For eg crosses and saltires have high correlation so we can keep one of the two and remove the other.
     The other two features are the second categorically encoded variable and the fifteenth encoded variable.

- **Automatic Feature Selection**
    1) **Variance Threshold**

       This function removes all the features below a threshold. It is unsupervised.Its done using the VarianceThreshold function from sklearn.feature_selection with threshold as the parameter.

       List of selected values: Landmass, zone, area, population, language, Religion, bars, stripes, colour, sunstar



```python
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import OneHotEncoder
import numpy as np
import pandas as pd

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

autosel = VarianceThreshold(threshold=(0.5*(0.5)))

Y = autosel.fit_transform(result_df)

print(Y)
```

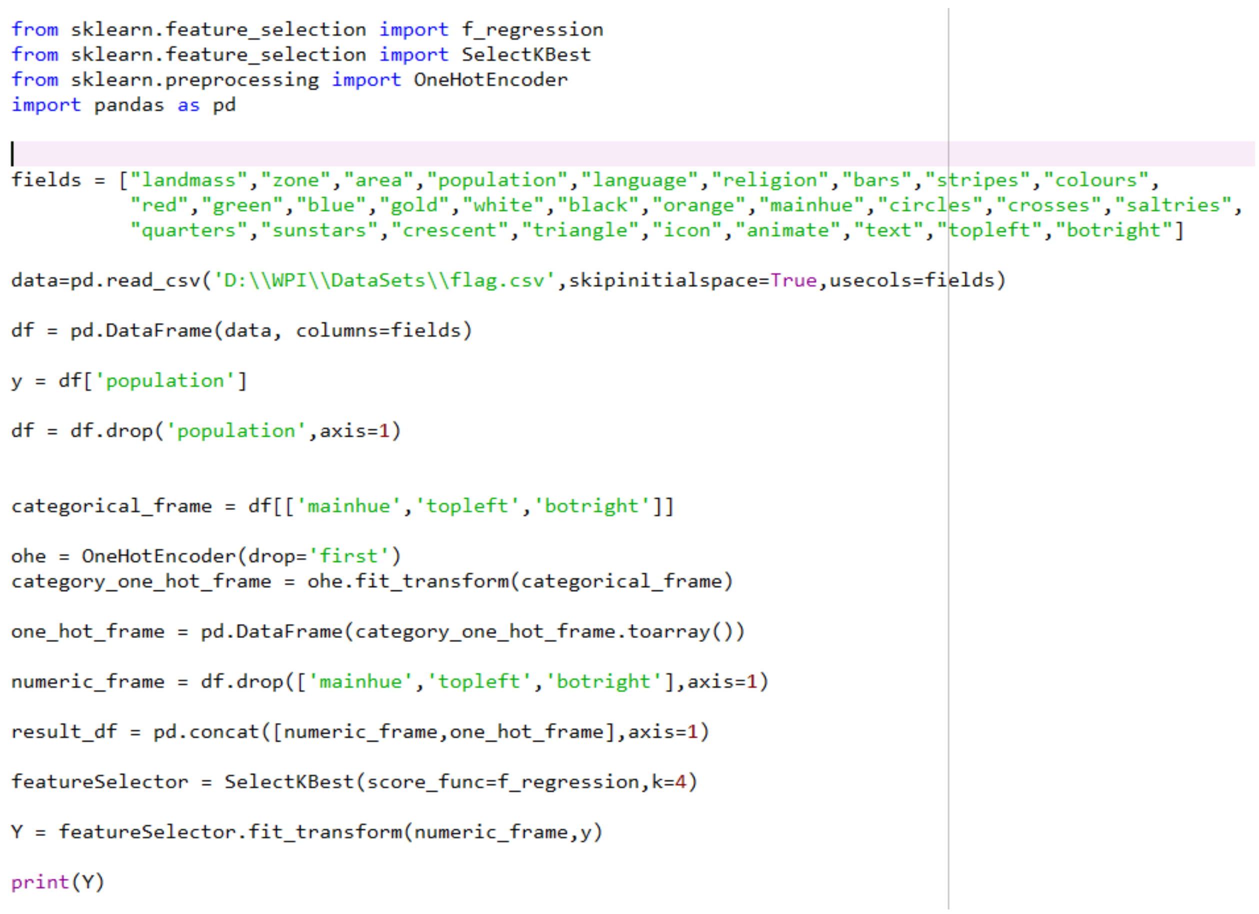
Fig 3.1 Automatic feature selection using variance threshold

2) **SelectKBest**

Selects the best features based on the scores, the features with high scores are retained.The features vary based on the score_func parameter.
Some of the values are discussed below.

- **f_regression**

When f_regression is selected as the scoring function with k=4 which represents the number of features to be selected, the following features are selected: **zone, area , religion and sunstar**



```
from sklearn.feature_selection import f_regression
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import OneHotEncoder
import pandas as pd


fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['population']

df = df.drop('population',axis=1)


categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

featureSelector = SelectKBest(score_func=f_regression,k=4)

Y = featureSelector.fit_transform(numeric_frame,y)

print(Y)
```

Fig 3.2 SelectKBest using f_regression as score function

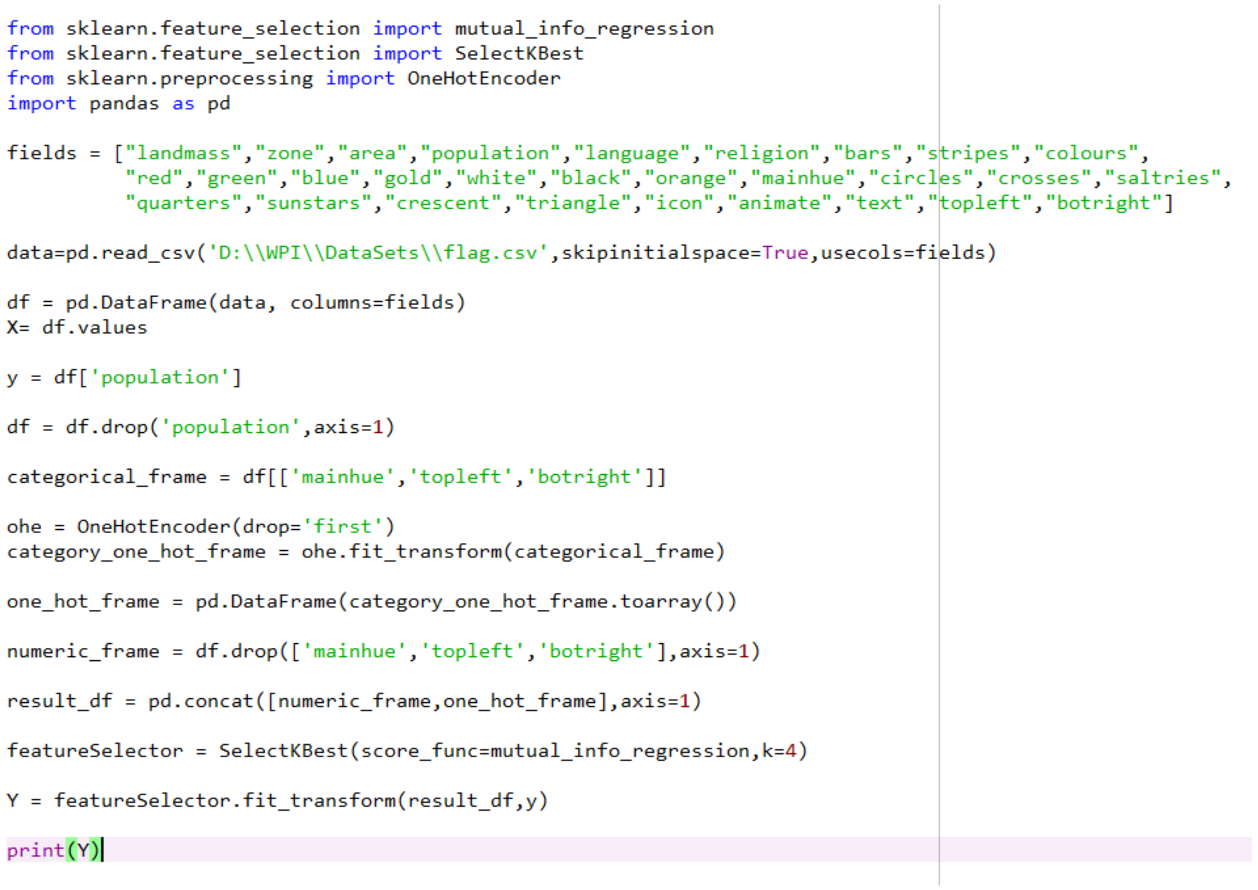
- **mutual_info_regression**

This function measures the dependence between two variables using entropy measures. When its 0 they are totally unrelated.

Scoring_func = mutual_info_regression

k=4 (number of features to select)

Outcome: landmass, area, lang, icon



```
from sklearn.feature_selection import mutual_info_regression
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)
X= df.values

y = df['population']

df = df.drop('population',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

featureSelector = SelectKBest(score_func=mutual_info_regression,k=4)

Y = featureSelector.fit_transform(result_df,y)

print(Y)
```

Fig 3.3 SelectKBest using mutual_info_regression as score function

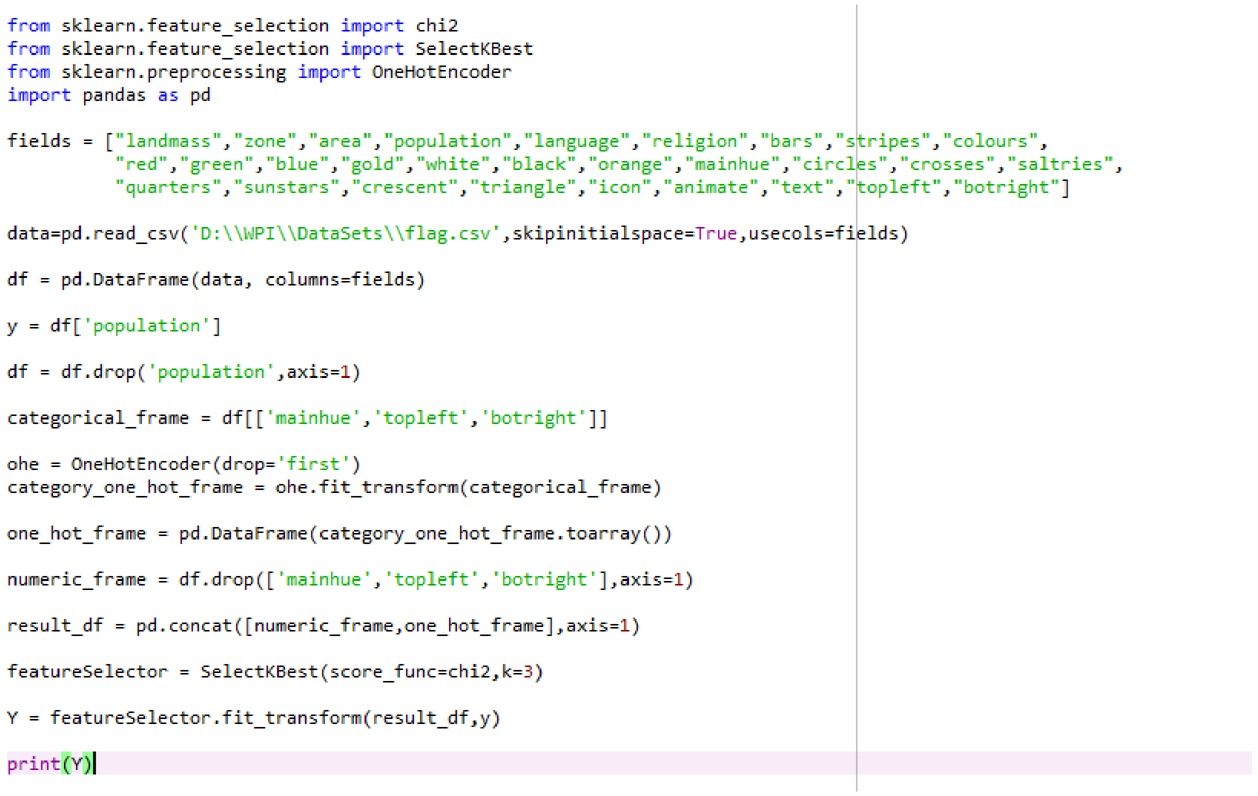
- **chi2**

This function selects the features with highest values after performing the chi-squared score.

Scoring_func = chi2

k=3 (number of features to select)

Outcome: landmass, stripes, sunstar



```python
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['population']

df = df.drop('population',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

featureSelector = SelectKBest(score_func=chi2,k=3)

Y = featureSelector.fit_transform(result_df,y)

print(Y)
```

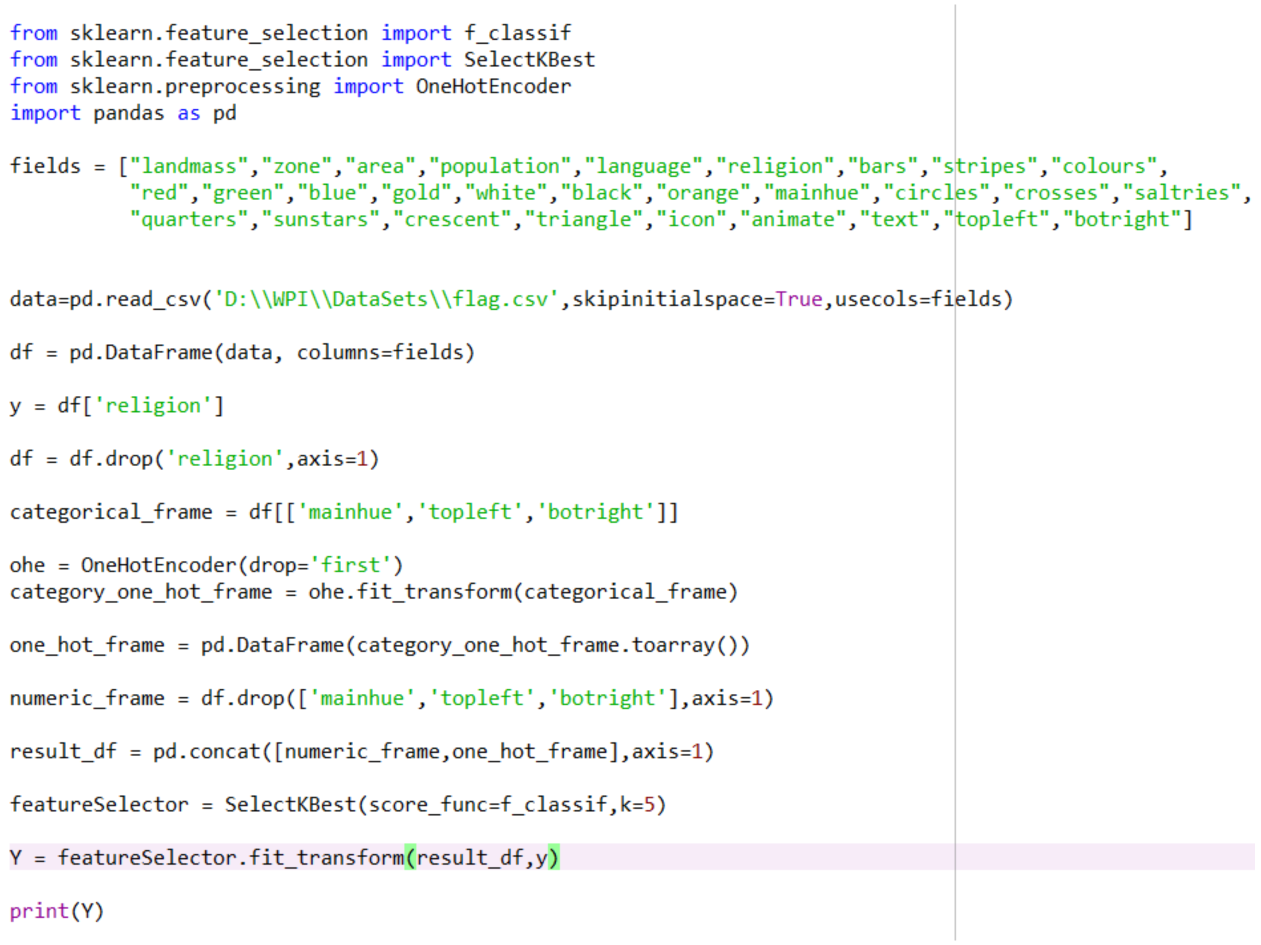
Fig 3.4 SelectKBest using chi2 as score function

- **F_classif**

Scoring_func = f_classis

k=5 (number of features to select)

Outcome: landmass, stripes, sunstar and two onehot encoded features

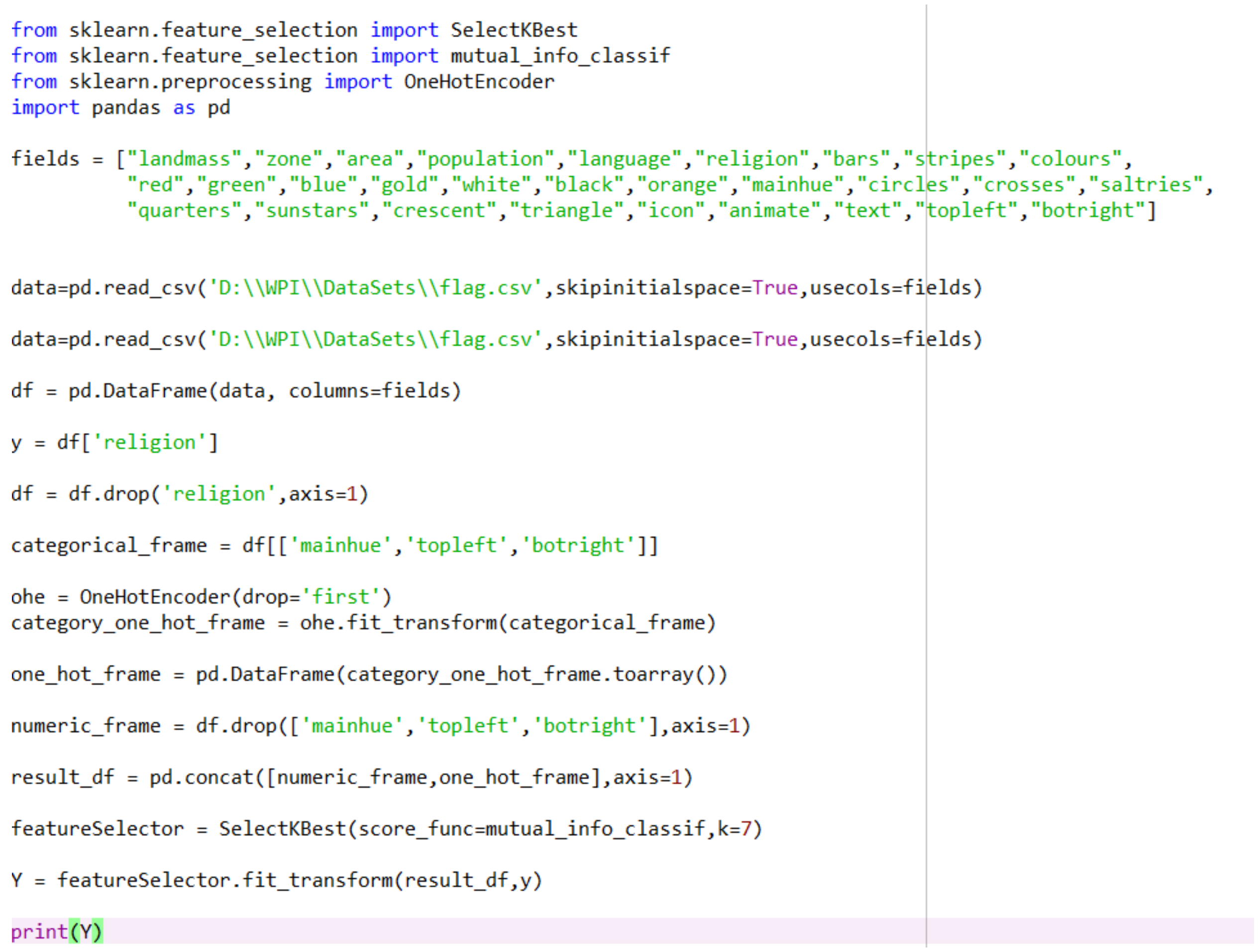


```python
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import SelectKBest
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

featureSelector = SelectKBest(score_func=f_classif,k=5)

Y = featureSelector.fit_transform(result_df,y)

print(Y)
```

Fig 3.5 SelectKBest using f_classic as score function

- **Mutual_info_classif**

Scoring_func = mutual_info_classif
k=7 (number of features to select)
Outcome: landmass, zone, area, language, red, blue and white



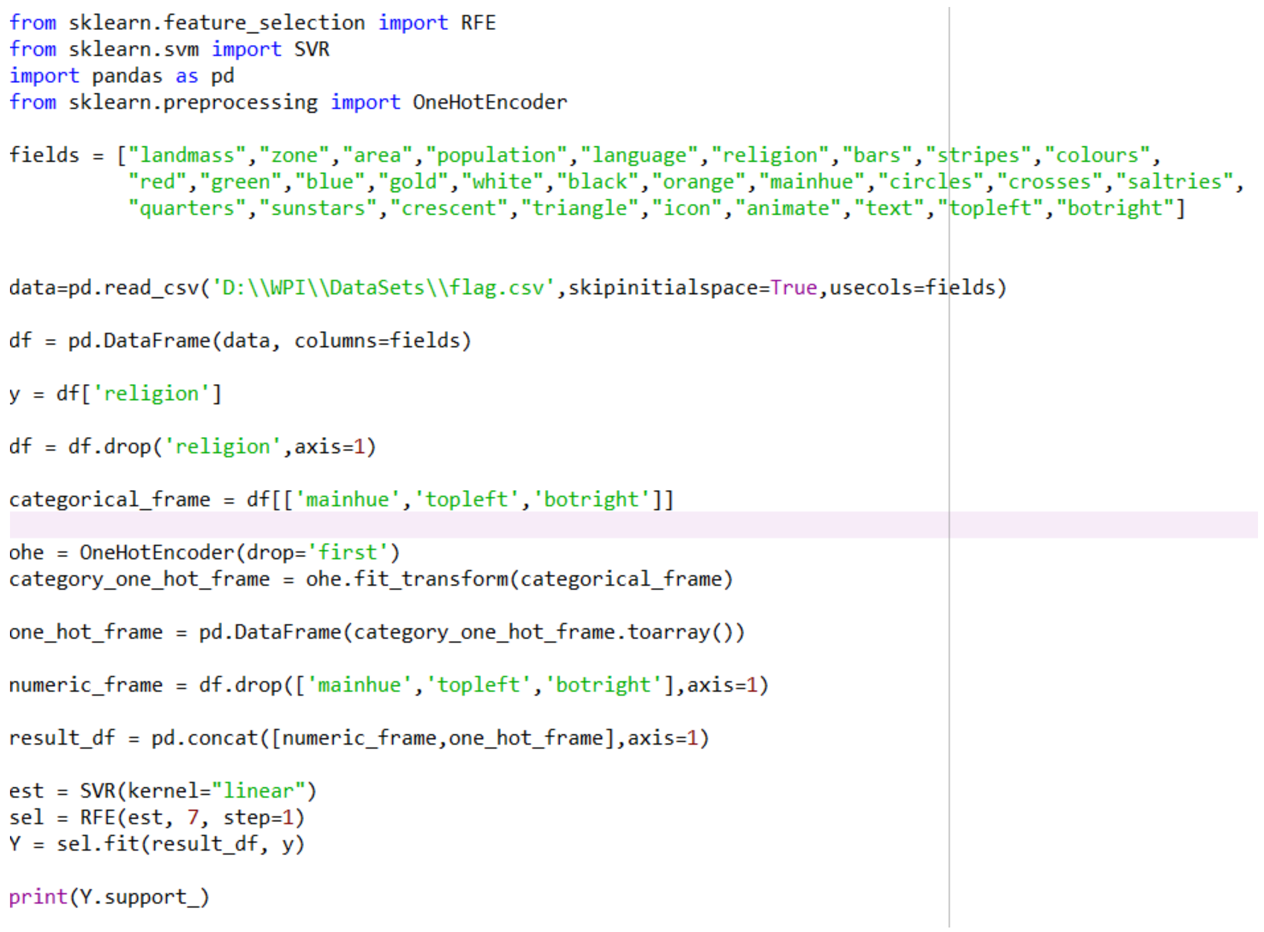
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import mutual_info_classif
from sklearn.preprocessing import OneHotEncoder
import pandas as pd

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

featureSelector = SelectKBest(score_func=mutual_info_classif,k=7)

Y = featureSelector.fit_transform(result_df,y)

print(Y)
```

Fig 3.6 SelectKBest using mutual_info_classif as score function

3) Recursive Feature Elimination

The purpose of this function is to iteratively reduce the number of features based on their weights.It first trains on the initial set and eliminates the least important feature.

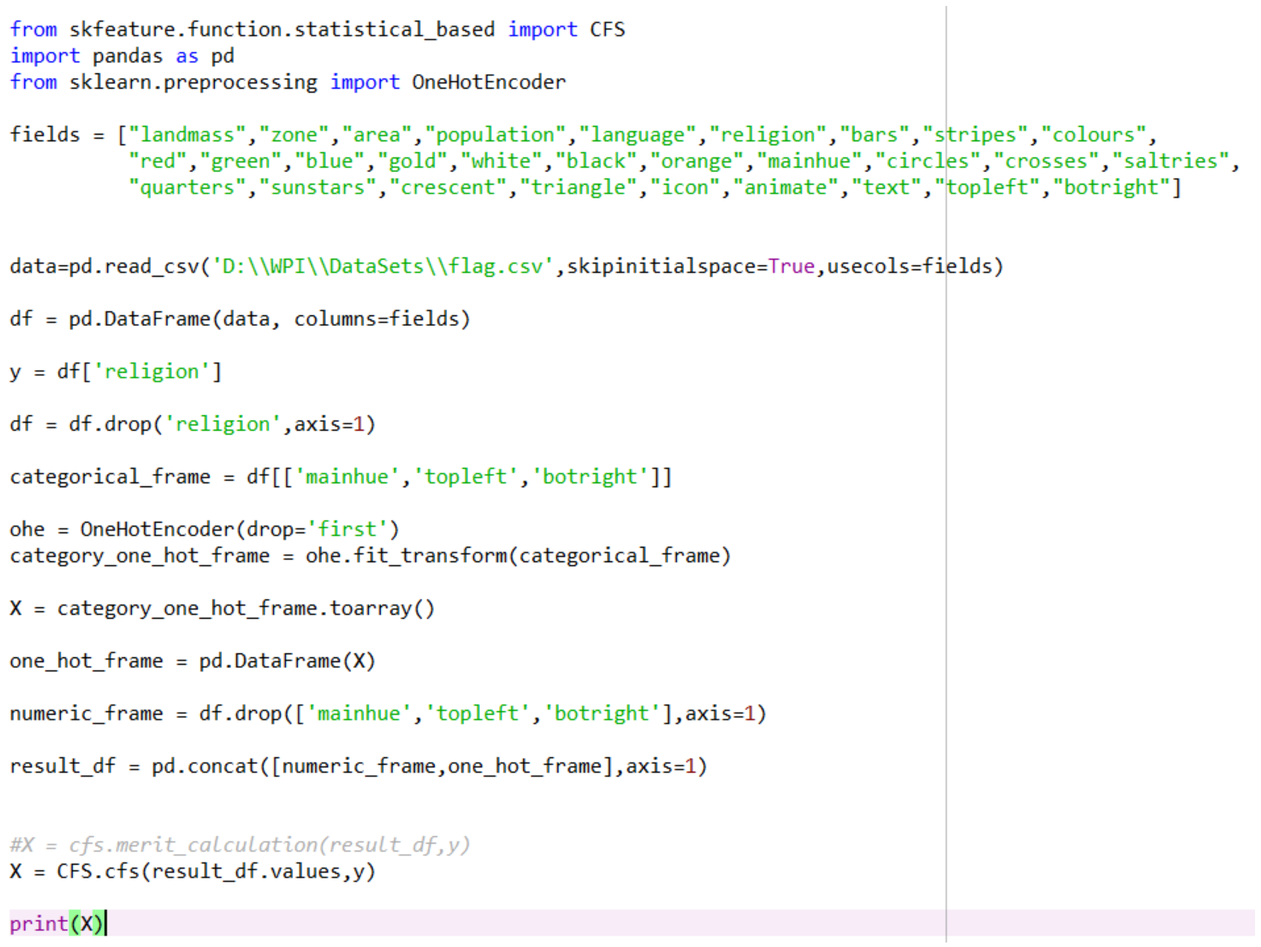
The parameters passed are the estimator and the number of features to select.



```python
from sklearn.feature_selection import RFE
from sklearn.svm import SVR
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

est = SVR(kernel="linear")
sel = RFE(est, 7, step=1)
Y = sel.fit(result_df, y)

print(Y.support_)
```

Fig 3.7 Code for recursive feature elimination

4) Correlation based Feature Selection



```python
from skfeature.function.statistical_based import CFS
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

X = category_one_hot_frame.toarray()

one_hot_frame = pd.DataFrame(X)

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)


#X = cfs.merit_calculation(result_df,y)
X = CFS.cfs(result_df.values,y)

print(X)
```

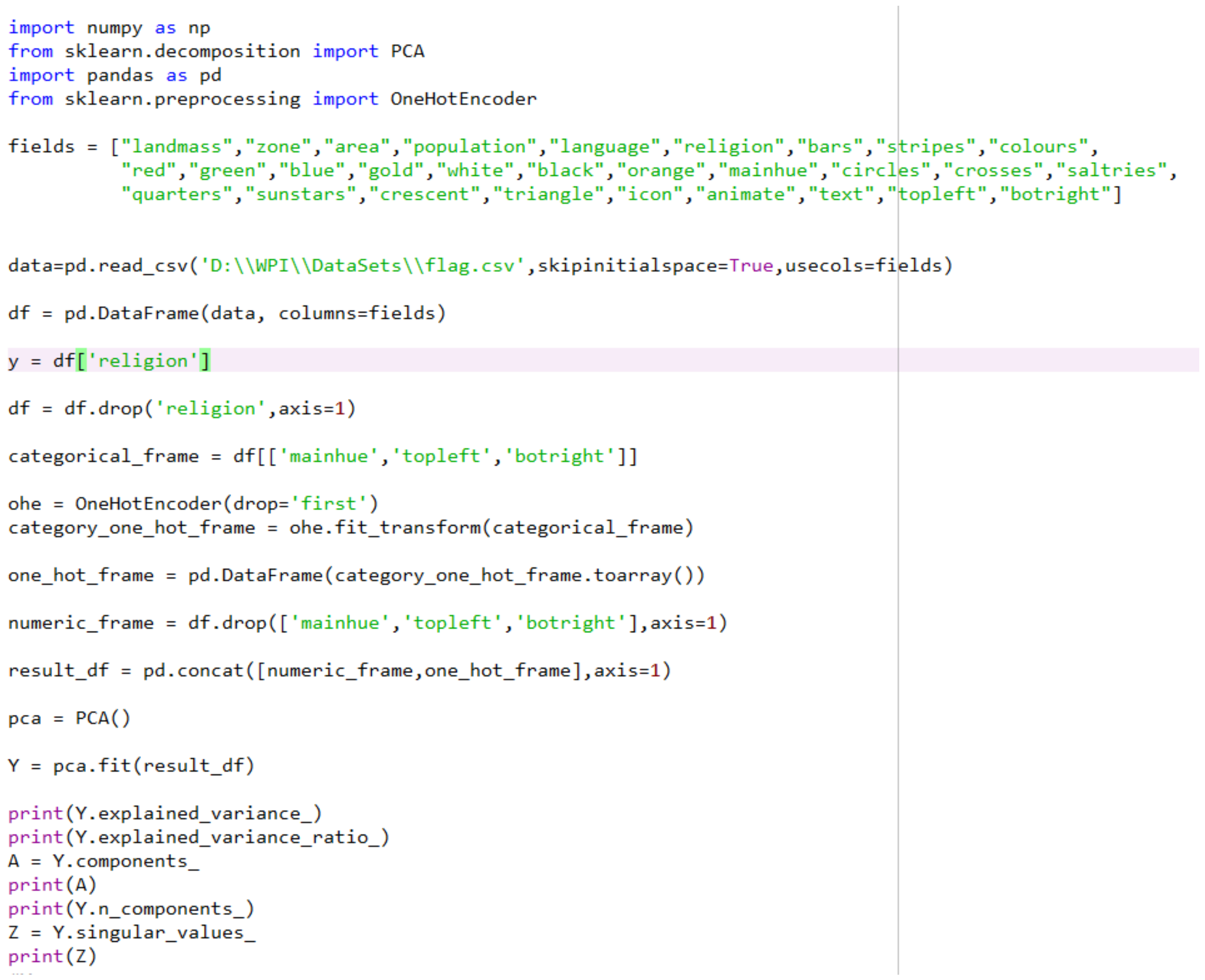
Fig 3.8 Code for Correlation based Feature Selection

From all the methods used above , it is evident that some of the features such as landmass, area and zone are present in almost all methods by increasing values of k where as some features like shapes never appear regardless of the high value of k This disparity shows not all the features are equally important.

4. Feature Extraction

Feature selection using PCA is used to reduce the dimensionality of the dataset by selecting the principle component features of the data.

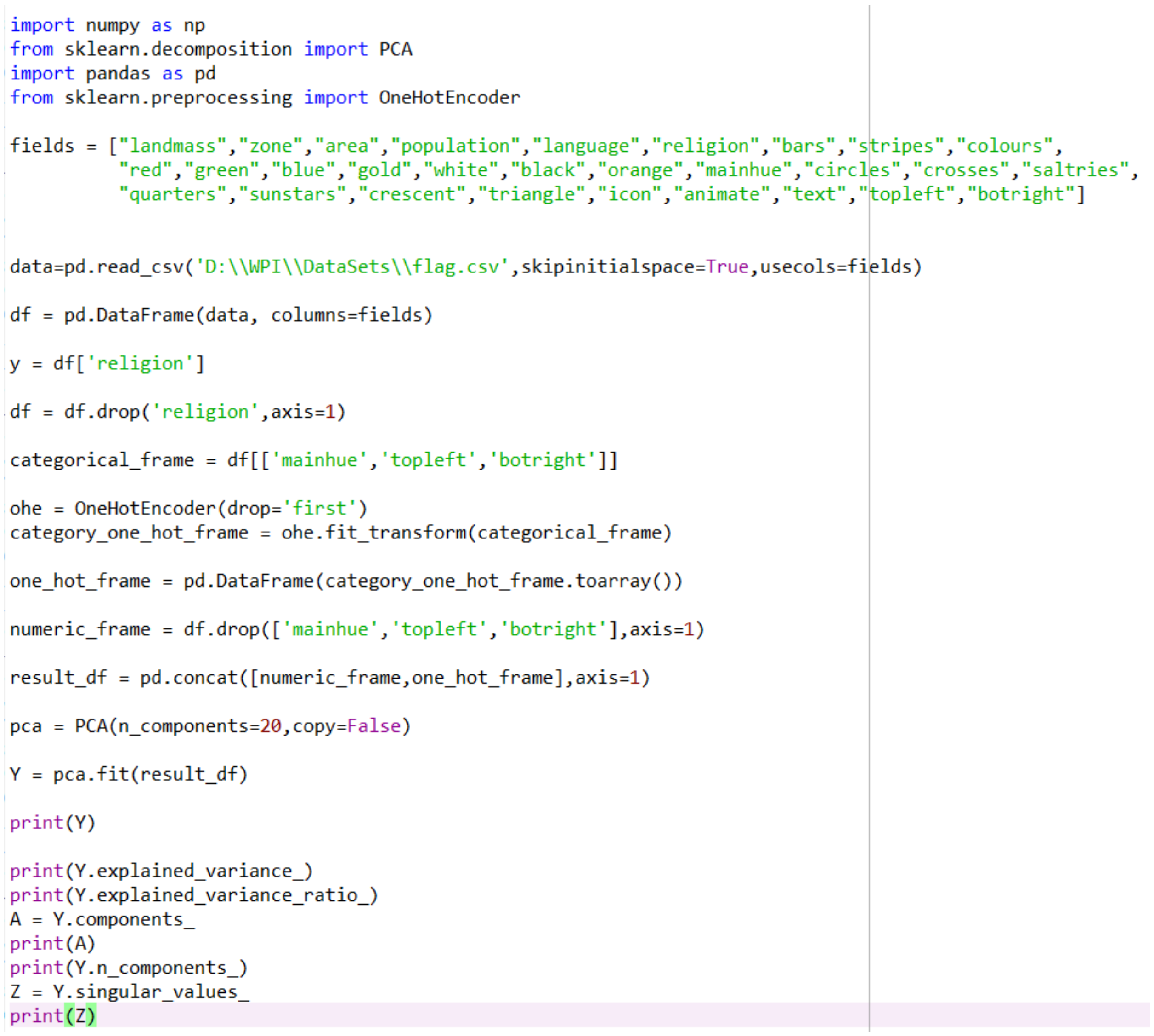
- **PCA with default parameters**
  While performing PCA with the default parameters n_components is not specified, which means all the components are selected by default.
  So the number of principle components returned is the number of features in the dataset(45). Explained variance ratio clearly shows that most of the variance is from a single attribute(area)



```
import numpy as np
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

pca = PCA()

Y = pca.fit(result_df)

print(Y.explained_variance_)
print(Y.explained_variance_ratio_)
A = Y.components_
print(A)
print(Y.n_components_)
Z = Y.singular_values_
print(Z)
```

Fig 3.9 Code for feature selection using PCA

- **PCA with reduced dimensions**
  Based on the explained variance and explained variance ratio the number of features to be selected can be determined.The number of features to be selected are the ones that contribute maximum variance.

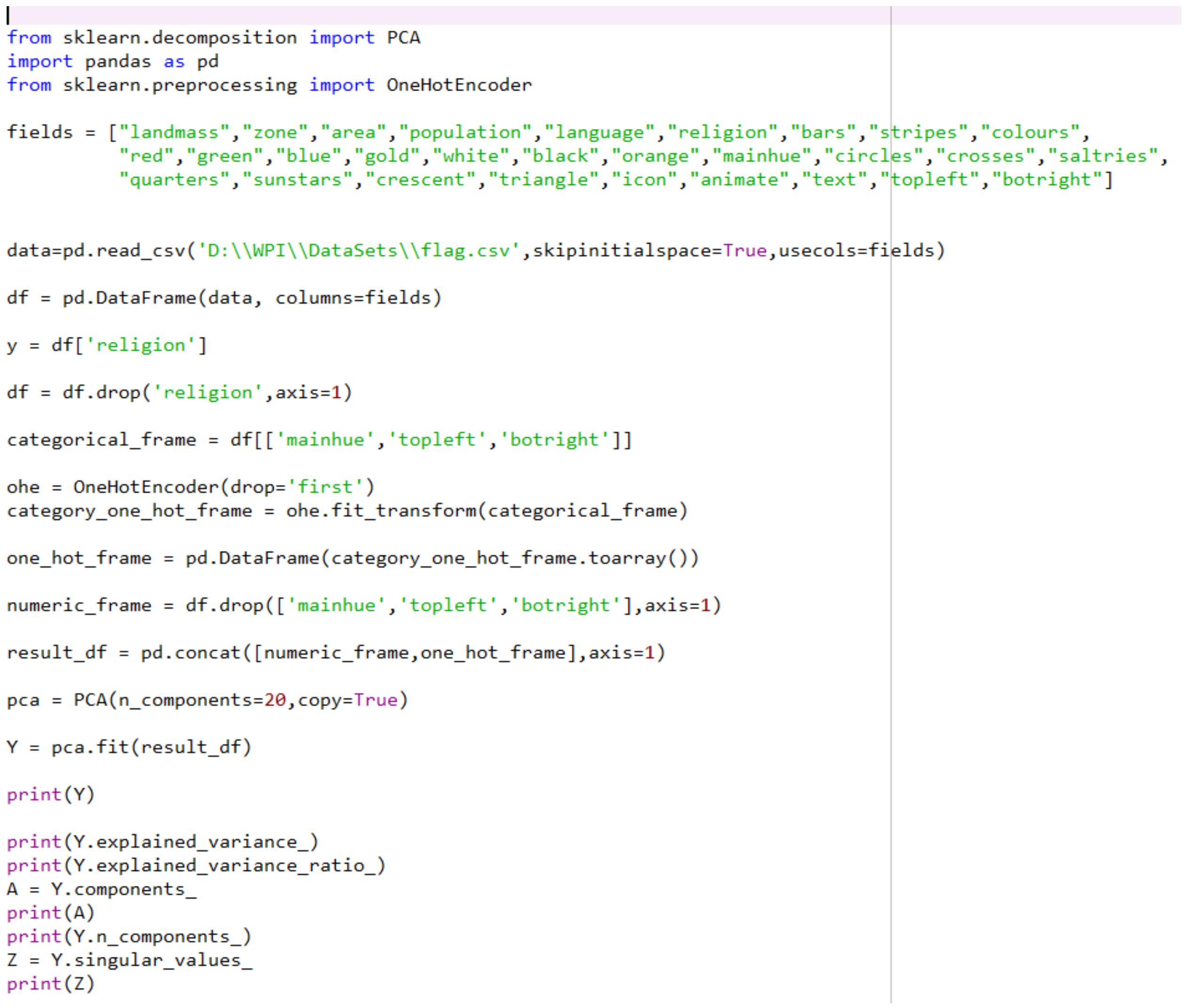


```
import numpy as np
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

pca = PCA(n_components=20,copy=False)

Y = pca.fit(result_df)

print(Y)

print(Y.explained_variance_)
print(Y.explained_variance_ratio_)
A = Y.components_
print(A)
print(Y.n_components_)
Z = Y.singular_values_
print(Z)
```

Fig 3.10 Code for feature selection using PCA with reduced dimensions

- **PCA with copy = True**

  Here the copy value is set to true, unlike the one discussed above.



```
from sklearn.decomposition import PCA
import pandas as pd
from sklearn.preprocessing import OneHotEncoder

fields = ["landmass","zone","area","population","language","religion","bars","stripes","colours",
          "red","green","blue","gold","white","black","orange","mainhue","circles","crosses","saltries",
          "quarters","sunstars","crescent","triangle","icon","animate","text","topleft","botright"]


data=pd.read_csv('D:\\WPI\\DataSets\\flag.csv',skipinitialspace=True,usecols=fields)

df = pd.DataFrame(data, columns=fields)

y = df['religion']

df = df.drop('religion',axis=1)

categorical_frame = df[['mainhue','topleft','botright']]

ohe = OneHotEncoder(drop='first')
category_one_hot_frame = ohe.fit_transform(categorical_frame)

one_hot_frame = pd.DataFrame(category_one_hot_frame.toarray())

numeric_frame = df.drop(['mainhue','topleft','botright'],axis=1)

result_df = pd.concat([numeric_frame,one_hot_frame],axis=1)

pca = PCA(n_components=20,copy=True)

Y = pca.fit(result_df)

print(Y)

print(Y.explained_variance_)
print(Y.explained_variance_ratio_)
A = Y.components_
print(A)
print(Y.n_components_)
Z = Y.singular_values_
print(Z)
```

Fig 3.11 Code for feature selection using PCA with copy value as true