Libraries

```python
# libraries to read data

import numpy as np
import pandas as pd

import regex

# pip install pandas-profiling
# from https://github.com/ydataai/pandas-profiling.git

from pandas_profiling import ProfileReport

# pip install lux-api

import lux

# libraries for making graphs

import seaborn as sns

import matplotlib.pyplot as plt
import matplotlib

# libraries for maps

import os
import json
import geopandas as gpd

# libraries for data analysis

import sklearn
from sklearn.linear_model import LinearRegression
```

Set directory

```python
import os
os.getcwd()
```

```
'/Users/elika_sinha/Documents/UCL/11. Dissertation/Term3'
```

```python
os.chdir("/Users/elika_sinha/Documents/UCL/11.
Dissertation/Term3/Datasets")
os.getcwd()
```

```
'/Users/elika_sinha/Documents/UCL/11. Dissertation/Term3/Datasets'
```

Final Dataset

```
# CIA data called directly
CIA = pd.read_csv('/Users/elika_sinha/Documents/UCL/11.
Dissertation/Term3/Datasets/Final_cleanData/CIA.csv')
CIA.info()
```

```
<class 'lux.core.frame.LuxDataFrame'>
RangeIndex: 850 entries, 0 to 849
Columns: 106 entries, OAs to CIA_Composite
dtypes: float64(105), object(1)
memory usage: 704.0+ KB
```

```
CIA.sample(5)
```

{"version_major":2,"version_minor":0,"model_id":"23eb4f3d174548ccbed36
f0b8fd45b66"}

{"version_major":2,"version_minor":0,"model_id":"df810839499b4fb19e856
5752784cb80"}

```
# CIA data called directly
CIA_Explore = pd.read_csv('/Users/elika_sinha/Documents/UCL/11.
Dissertation/Term3/Datasets/Final_cleanData/CIA_Explore.csv')
CIA_Explore.info()
```

```
<class 'lux.core.frame.LuxDataFrame'>
RangeIndex: 783 entries, 0 to 782
Columns: 112 entries, OAs to CIA_Composite
dtypes: float64(107), int64(3), object(2)
memory usage: 685.2+ KB
```

Decision Tree Regressor

```
from sklearn.model_selection import train_test_split
import statsmodels.api as sm
plt.style.use('ggplot') # specifies that graphs should use ggplot
styling
%matplotlib inline
```

A. Whole DATA

```
# setting test and train split
random_state_split = 100
train_x1, test_x1, train_y1, test_y1 =
train_test_split(CIA.drop(['OAs', 'CIA_Composite'], axis = 1),
CIA.CIA_Composite, random_state=random_state_split)

# to print split
print(train_x1.shape)
print(train_y1.shape)
print(test_x1.shape)
print(test_y1.shape)
```

```
# checking the test and train index

print(train_x1.index.identical(train_y1.index))
print(test_x1.index.identical(test_y1.index))
```

```
(637, 104)
(637,)
(213, 104)
(213,)
True
True
```

```
from sklearn.tree import DecisionTreeRegressor
reg_tree1 = DecisionTreeRegressor(random_state=0)
reg_tree1.fit(train_x1, train_y1)
```

```
DecisionTreeRegressor(random_state=0)
```

```
print("R2 on the training data:")
print(reg_tree1.score(X=train_x1, y=train_y1))
print("R2 on the testing data:")
print(reg_tree1.score(X=test_x1, y=test_y1))
```

```
R2 on the training data:
1.0
R2 on the testing data:
0.9188679674668143
```

```
from sklearn.metrics import mean_squared_error
print("RMSE on the training data:")
print(mean_squared_error(train_y1, reg_tree1.predict(train_x1),
squared=False))
print("RMSE on the testing data:")
print(mean_squared_error(test_y1, reg_tree1.predict(test_x1),
squared=False))
```

```
RMSE on the training data:
0.0
RMSE on the testing data:
292305.12542232283
```

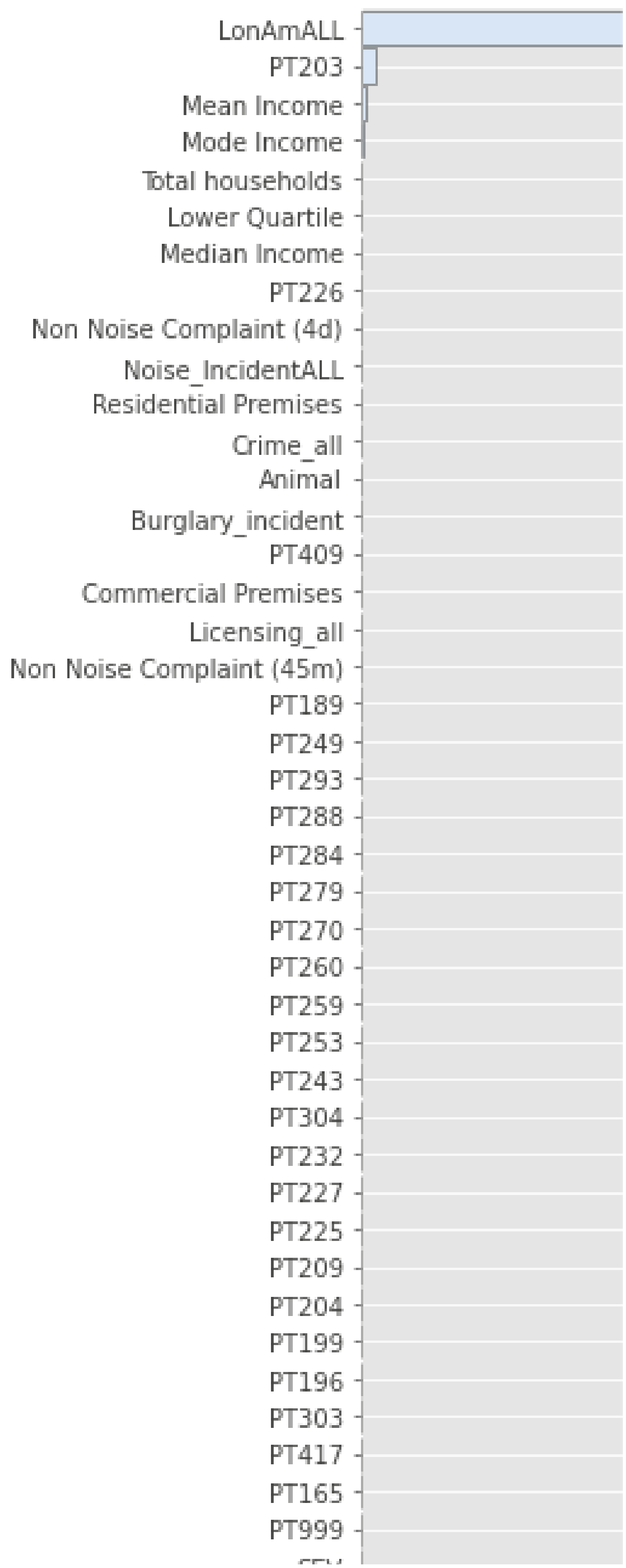
```
print("Depth of the regression tree:
{}".format(reg_tree1.get_depth()))
print("Number of nodes of this tree:
{}".format(reg_tree1.get_n_leaves()))
```

```
Depth of the regression tree: 20
Number of nodes of this tree: 608
```

```
import rfpimp
imp1 = rfpimp.importances(reg_tree1, test_x1, test_y1) # permutation
```

```
print(imp1)
viz = rfpimp.plot_importances(imp1)
viz.view()
```

```
                   Importance
Feature
LonAmALL             1.204662
PT203                0.066810
Mean Income          0.023387
Mode Income          0.009080
Total households     0.001466
...                       ...
Street              -0.000003
Building Site       -0.000004
Violence_incident   -0.000076
Disorder_incident   -0.000458
Fraud_incident      -0.001731

[104 rows x 1 columns]
```



LonAmALL
PT203
Mean Income
Mode Income
Total households
Lower Quartile
Median Income
PT226
Non Noise Complaint (4d)
Noise_IncidentALL
Residential Premises
Crime_all
Animal
Burglary_incident
PT409
Commercial Premises
Licensing_all
Non Noise Complaint (45m)
PT189
PT249
PT293
PT288
PT284
PT279
PT270
PT260
PT259
PT253
PT243
PT304
PT232
PT227
PT225
PT209
PT204
PT199
PT196
PT303
PT417
PT165
PT999

B. SET SELECTED GRID

```
CIA_Grid =
CIA.filter(['CIA_Composite','Licensing_all','Crime_all','LonAmALL',
'Mean Income', 'Total households', 'Noise_IncidentALL'], axis=1)

CIA_Grid.info()

<class 'lux.core.frame.LuxDataFrame'>
RangeIndex: 850 entries, 0 to 849
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   CIA_Composite      850 non-null    float64
 1   Licensing_all      850 non-null    float64
 2   Crime_all          850 non-null    float64
 3   LonAmALL           850 non-null    float64
 4   Mean Income        850 non-null    float64
 5   Total households   850 non-null    float64
 6   Noise_IncidentALL  850 non-null    float64
dtypes: float64(7)
memory usage: 46.6 KB

# setting test and train split
random_state_split = 100
train_x2, test_x2, train_y2, test_y2 =
train_test_split(CIA_Grid.drop(['CIA_Composite'], axis = 1),
CIA_Grid.CIA_Composite, random_state=random_state_split)

# to print split
print(train_x2.shape)
print(train_y2.shape)
print(test_x2.shape)
print(test_y2.shape)

# checking the test and train index

print(train_x2.index.identical(train_y2.index))
print(test_x2.index.identical(test_y2.index))

(637, 6)
(637,)
(213, 6)
(213,)
True
True

reg_tree2 = DecisionTreeRegressor(random_state=0)
reg_tree2.fit(train_x2, train_y2)

DecisionTreeRegressor(random_state=0)
```

```
print("R2 on the training data:")
print(reg_tree2.score(X=train_x2, y=train_y2))
print("R2 on the testing data:")
print(reg_tree2.score(X=test_x2, y=test_y2))
```

```
R2 on the training data:
1.0
R2 on the testing data:
0.99256578543714
```

```
print("RMSE on the training data:")
print(mean_squared_error(train_y2, reg_tree2.predict(train_x2),
squared=False))
print("RMSE on the testing data:")
print(mean_squared_error(test_y2, reg_tree2.predict(test_x2),
squared=False))
```

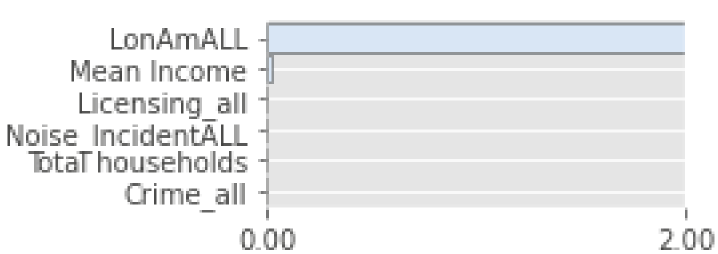
```
RMSE on the training data:
0.0
RMSE on the testing data:
88482.58422542024
```

```
print("Depth of the regression tree:
{}".format(reg_tree2.get_depth()))
print("Number of nodes of this tree:
{}".format(reg_tree2.get_n_leaves()))
```

```
Depth of the regression tree: 19
Number of nodes of this tree: 608
```

```
imp3 = rfpimp.importances(reg_tree2, test_x2, test_y2) # permutation
print(imp3)
viz = rfpimp.plot_importances(imp3)
viz.view()
```

```
                   Importance
Feature
LonAmALL             1.994014
Mean Income          0.018150
Licensing_all        0.000048
Noise_IncidentALL   -0.000014
Total households    -0.000103
Crime_all           -0.000323
```



Random Forest Regressor

A. WHOLE DATA

```
from sklearn.ensemble import RandomForestRegressor
reg_random_forest1 = RandomForestRegressor(random_state=0)
reg_random_forest1.fit(train_x1, train_y1)
```

```
RandomForestRegressor(random_state=0)
```

```
print("R2 on the training data:")
print(reg_random_forest1.score(X=train_x1, y=train_y1))
print("R2 on the testing data:")
print(reg_random_forest1.score(X=test_x1, y=test_y1))
```

```
R2 on the training data:
0.9790022406843535
R2 on the testing data:
0.9436493763518087
```

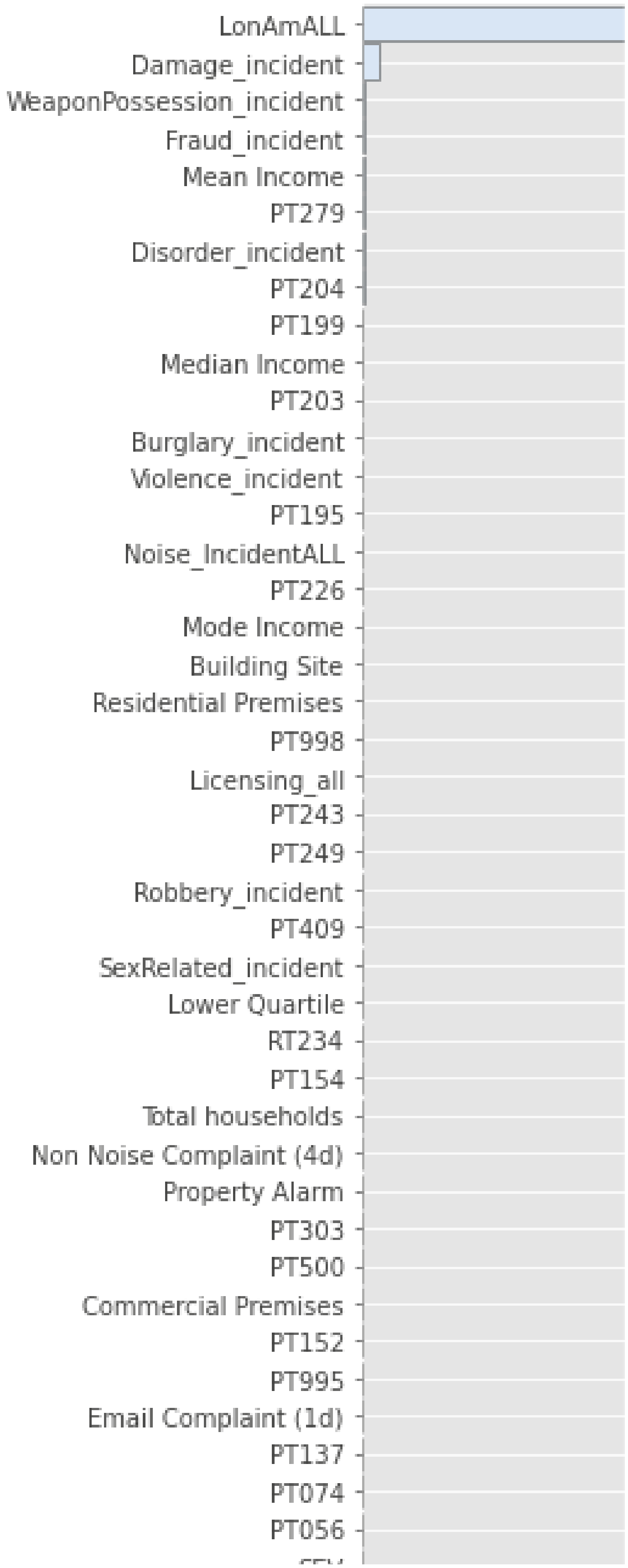
```
from sklearn.metrics import mean_squared_error
print("RMSE on the training data:")
print(mean_squared_error(train_y1,
reg_random_forest1.predict(train_x1), squared=False))
print("RMSE on the testing data:")
print(mean_squared_error(test_y1, reg_random_forest1.predict(test_x1),
squared=False))
```

```
RMSE on the training data:
110148.13829877623
RMSE on the testing data:
243606.91721743369
```

```
imp2 = rfpimp.importances(reg_random_forest1, test_x1, test_y1) #
permutation
print(imp2)
viz = rfpimp.plot_importances(imp2)
viz.view()
```

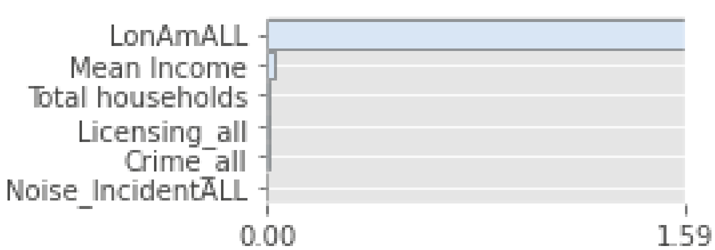
```
                           Importance
Feature
LonAmALL                     1.027970
Damage_incident              0.061080
WeaponPossession_incident    0.008264
Fraud_incident               0.006391
Mean Income                  0.006119
...                               ...
PT234                       -0.000311
Non Noise Complaint (45m)   -0.000565
Animal                      -0.001291
PT138                       -0.001710
PT227                       -0.002848
```

[104 rows x 1 columns]


LonAmALL
Damage_incident
WeaponPossession_incident
Fraud_incident
Mean Income
PT279
Disorder_incident
PT204
PT199
Median Income
PT203
Burglary_incident
Violence_incident
PT195
Noise_IncidentALL
PT226
Mode Income
Building Site
Residential Premises
PT998
Licensing_all
PT243
PT249
Robbery_incident
PT409
SexRelated_incident
Lower Quartile
RT234
PT154
Total households
Non Noise Complaint (4d)
Property Alarm
PT303
PT500
Commercial Premises
PT152
PT995
Email Complaint (1d)
PT137
PT074
PT056

B. SET SELECTED GRID

```
reg_random_forest2 = RandomForestRegressor(random_state=0)
reg_random_forest2.fit(train_x2, train_y2)

RandomForestRegressor(random_state=0)

print("R2 on the training data:")
print(reg_random_forest2.score(X=train_x2, y=train_y2))
print("R2 on the testing data:")
print(reg_random_forest2.score(X=test_x2, y=test_y2))

R2 on the training data:
0.9856671477193147
R2 on the testing data:
0.976667490800419

print("RMSE on the training data:")
print(mean_squared_error(train_y2,
reg_random_forest2.predict(train_x2), squared=False))
print("RMSE on the testing data:")
print(mean_squared_error(test_y2, reg_random_forest2.predict(test_x2),
squared=False))

RMSE on the training data:
91003.26978124736
RMSE on the testing data:
156754.84587223164

imp4 = rfpimp.importances(reg_random_forest2, test_x2, test_y2) #
permutation
print(imp4)
viz = rfpimp.plot_importances(imp4)
viz.view()

                   Importance
Feature
LonAmALL             1.589659
Mean Income          0.023187
Total households     0.003479
Licensing_all        0.001947
Crime_all            0.001347
Noise_IncidentALL   -0.000248
```



```
# create a list of models for whole dataset
list_name_models = ['DETREE', 'RF']
# use the models from above
list_reg_models = [reg_tree1, reg_random_forest1,]

dict_models = dict()

for name, model in zip(list_name_models, list_reg_models):
    if name == 'DETREE':
        dict_models[name] = [model.score(train_x1, train_y1),
model.score(test_x1, test_y1)]
    else:
        dict_models[name] = [model.score(train_x1, train_y1),
model.score(test_x1, test_y1)]

df_models1 = pd.DataFrame.from_dict(dict_models, orient='index',
columns=['R2_train_data', 'R2_test_data'])
print(df_models1)
```

```
        R2_train_data  R2_test_data
DETREE       1.000000      0.918868
RF           0.979002      0.943649
```

```
# create a list of models for set grid dataset
list_name_models = ['DETREE', 'RF']
# use the models from above
list_reg_models = [reg_tree2, reg_random_forest2,]

dict_models = dict()

for name, model in zip(list_name_models, list_reg_models):
    if name == 'DETREE':
        dict_models[name] = [model.score(train_x2, train_y2),
model.score(test_x2, test_y2)]
    else:
        dict_models[name] = [model.score(train_x2, train_y2),
model.score(test_x2, test_y2)]

df_models2 = pd.DataFrame.from_dict(dict_models, orient='index',
columns=['R2_train_data', 'R2_test_data'])
print(df_models2)
```

```
        R2_train_data  R2_test_data
DETREE       1.000000      0.992566
RF           0.985667      0.976667
```

```
from prettytable import PrettyTable

# Specify the Column Names while initializing the Table
Results1 = PrettyTable([" ", "CIA"])
```

```
# Add rows
Results1.add_row([" ", " "])

Results1.add_row(["Most important feature_DeTree", "Ambulance
Incident, Licensing-PT203"])
Results1.add_row(["R2 Value_train data_DeTree",
{reg_tree1.score(X=train_x1, y=train_y1)}])
Results1.add_row(["R2 Value_test data_DeTree",
{reg_tree1.score(X=test_x1, y=test_y1)}])
Results1.add_row(["Mean Squared Error_train_DeTree",
{(mean_squared_error(train_y1, reg_tree1.predict(train_x1),
squared=False))}])
Results1.add_row(["Mean Squared Error_test_DeTree",
{(mean_squared_error(test_y1, reg_tree1.predict(test_x1),
squared=False))}])

Results1.add_row([" ", " "])

Results1.add_row(["Most important feature_RF", "Ambulance Incident,
Damage Incident"])
Results1.add_row(["R2 Value_train data_RF",
{(reg_random_forest1.score(X=train_x1, y=train_y1))}])
Results1.add_row(["R2 Value_test data_RF",
{(reg_random_forest1.score(X=test_x1, y=test_y1))}])
Results1.add_row(["Mean Squared Error_train_RF",
{mean_squared_error(train_y1, reg_random_forest1.predict(train_x1),
squared=False)}])
Results1.add_row(["Mean Squared Percentage Error_test_RF",
{mean_squared_error(test_y1, reg_random_forest1.predict(test_x1),
squared=False)}])

print(Results1)
```

```
+-------------------------------------
+------------------------------------+
|                                      |                 CIA
|
+-------------------------------------
+------------------------------------+
|                                      |
|
|     Most important feature_DeTree     | Ambulance Incident,
Licensing-PT203 |
|       R2 Value_train data_DeTree      |                {1.0}
|
|       R2 Value_test data_DeTree       |         {0.9188679674668143}
|
|    Mean Squared Error_train_DeTree    |                {0.0}
|
```

```
|     Mean Squared Error_test_DeTree    |         {292305.12542232283}
|
|                                       |
|
|       Most important feature_RF       | Ambulance Incident, Damage
Incident |
|         R2 Value_train data_RF        |         {0.9790022406843535}
|
|         R2 Value_test data_RF         |         {0.9436493763518087}
|
|      Mean Squared Error_train_RF      |         {110148.13829877623}
|
| Mean Squared Percentage Error_test_RF |         {243606.91721743369}
|
+---------------------------------------
+---------------------------------------+

# Specify the Column Names while initializing the Table
Results2 = PrettyTable([" ", "CIA"])

# Add rows
Results2.add_row([" ", " "])

Results2.add_row(["Most important feature_DeTree", "Ambulance
Incident, Income"])
Results2.add_row(["R2 Value_train data_DeTree",
{reg_tree2.score(X=train_x2, y=train_y2)}])
Results2.add_row(["R2 Value_test data_DeTree",
{reg_tree2.score(X=test_x2, y=test_y2)}])
Results2.add_row(["Mean Squared Error_train_DeTree",
{(mean_squared_error(train_y2, reg_tree2.predict(train_x2),
squared=False))}])
Results2.add_row(["Mean Squared Error_test_DeTree",
{(mean_squared_error(test_y2, reg_tree2.predict(test_x2),
squared=False))}])

Results2.add_row([" ", " "])

Results2.add_row(["Most important feature_RF", "Ambulance Incident,
Licensing"])
Results2.add_row(["R2 Value_train data_RF",
{(reg_random_forest2.score(X=train_x2, y=train_y2))}])
Results2.add_row(["R2 Value_test data_RF",
{(reg_random_forest2.score(X=test_x2, y=test_y2))}])
Results2.add_row(["Mean Squared Error_train_RF",
{mean_squared_error(train_y2, reg_random_forest2.predict(train_x2),
squared=False)}])
Results2.add_row(["Mean Squared Percentage Error_test_RF",
{mean_squared_error(test_y2, reg_random_forest2.predict(test_x2),
```

```
squared=False)}])

print(Results2)
```

```
+--------------------------------------
+-----------------------------+
|                                      |              CIA
|
+--------------------------------------
+-----------------------------+
|                                      |
|
|     Most important feature_DeTree     |   Ambulance Incident, Income
|
|       R2 Value_train data_DeTree      |             {1.0}
|
|       R2 Value_test data_DeTree       |       {0.99256578543714}
|
|    Mean Squared Error_train_DeTree    |             {0.0}
|
|     Mean Squared Error_test_DeTree    |      {88482.58422542024}
|
|                                      |
|
|       Most important feature_RF       | Ambulance Incident,
Licensing |
|         R2 Value_train data_RF        |      {0.9856671477193147}
|
|         R2 Value_test data_RF         |      {0.976667490800419}
|
|      Mean Squared Error_train_RF      |      {91003.26978124736}
|
| Mean Squared Percentage Error_test_RF |      {156754.84587223164}
|
+--------------------------------------
+-----------------------------+
```

PCA

```
# Machine Learning
from sklearn import metrics
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.pipeline import Pipeline
from sklearn.pipeline import FeatureUnion
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.decomposition import IncrementalPCA
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
```

```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Plotting
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Whole dataset

```python
# drop 'OAs' to create X - the main geographical element
X1 = CIA.drop(['OAs', 'CIA_Composite'], axis=1).values

# the name list of independent variables
list_var_X1 = CIA.columns.tolist()
list_var_X1.remove('OAs')
#list_var_X1.remove('geometry')
list_var_X1.remove('CIA_Composite')

y1 = CIA.loc[:,['OAs', 'CIA_Composite']].values

from sklearn.preprocessing import StandardScaler
X1_std = StandardScaler().fit_transform(X1)

from sklearn.decomposition import PCA
rand_st_int = 10
pca1 = PCA(random_state=rand_st_int)
# fit the components
X_new_components1 = pca1.fit_transform(X1)

print(list_var_X1)
```

```
['Total households', 'Mean Income', 'Median Income', 'Mode Income',
'Lower Quartile', 'LonAmALL', 'Damage_incident', 'Burglary_incident',
'Disorder_incident', 'Fraud_incident', 'Robbery_incident',
'SexRelated_incident', 'Violence_incident',
'WeaponPossession_incident', 'Crime_all', 'Noise_IncidentALL',
'Animal', 'Building Site', 'Commercial Premises', 'Email Complaint
(1d)', 'Formal complaints', 'Non Noise Complaint (45m)', 'Non Noise
Complaint (4d)', 'Proactive Noise', 'Property Alarm', 'Residential
Premises', 'Street', 'VIP complaint', 'GACLGE', 'GAVESS', 'LIMSTL',
'LIPSL', 'PT011', 'PT019', 'PT031', 'PT049', 'PT056', 'PT057',
'PT060', 'PT061', 'PT062', 'PT065', 'PT070', 'PT074', 'PT075',
'PT082', 'PT086', 'PT100', 'PT104', 'PT106', 'PT122', 'PT135',
'PT137', 'PT138', 'PT139', 'PT140', 'PT152', 'PT154', 'PT155',
'PT165', 'PT189', 'PT195', 'PT196', 'PT199', 'PT203', 'PT204',
'PT209', 'PT225', 'PT226', 'PT227', 'PT232', 'PT234', 'PT243',
'PT249', 'PT253', 'PT259', 'PT260', 'PT270', 'PT279', 'PT284',
'PT288', 'PT293', 'PT303', 'PT304', 'PT409', 'PT417', 'PT437',
'PT439', 'PT442', 'PT500', 'PT504', 'PT508', 'PT993', 'PT995',
```

```
'PT998', 'PT999', 'RT061', 'RT199', 'RT442', 'RT226', 'RT234',
'RT303', 'SEV', 'Licensing_all']
```

```
df_PrincipleComp1 = pd.DataFrame(pca1.components_, columns =
list_var_X1)
df_PrincipleComp1
```

{"version_major":2,"version_minor":0,"model_id":"624e949ff0dd4f9cb4ba4
94b68a40729"}

{"version_major":2,"version_minor":0,"model_id":"e5e42b0f7ec44d76a3354
0b051bb62c8"}

```
df_PrincipleComp1.info()
```

```
<class 'lux.core.frame.LuxDataFrame'>
RangeIndex: 104 entries, 0 to 103
Columns: 104 entries, Total households to Licensing_all
dtypes: float64(104)
memory usage: 84.6 KB
```

```
print(df_PrincipleComp1)
```

```
     Total households   Mean Income  Median Income   Mode Income  \
0            0.000009  1.315619e-03   1.121764e-03  3.688047e-03
1           -0.000564 -3.810806e-01  -3.593581e-01 -8.184402e-01
2            0.001236  5.638265e-01   5.044154e-01 -5.742712e-01
3            0.016656  6.422260e-01  -2.560167e-01  1.877961e-02
4            0.002744 -7.076987e-02   1.157051e-02 -1.883069e-04
..                ...           ...            ...           ...
99           0.000000 -5.689351e-17   1.547255e-16  5.827451e-18
100          0.000000  7.803545e-17  -1.794623e-16 -4.897021e-18
101          0.000000 -1.384899e-18   3.116636e-18 -1.718613e-19
102          0.000000  2.611670e-23  -1.044111e-22  3.137987e-24
103          0.000000 -1.972028e-16   5.290267e-16 -2.593525e-18

     Lower Quartile      LonAmALL  Damage_incident  Burglary_incident
\
0      6.827289e-04  9.999909e-01     2.915862e-05       2.612658e-05

1     -2.361731e-01  4.084497e-03    -2.113598e-05      -7.861451e-05

2      3.128134e-01  5.921352e-04     1.663442e-04       2.620105e-04

3     -7.119807e-01 -2.631768e-04     2.790686e-03       3.940450e-03

4      9.617322e-02 -9.757615e-04     1.167085e-02       1.786737e-02

..              ...           ...              ...                ...

99    -1.356456e-16  6.375711e-19     6.144110e-03       6.144110e-03
```

```
100    1.511652e-16  3.672929e-18    -4.669138e-03      -4.669138e-03

101   -3.066249e-18  1.826966e-19     1.614607e-03       1.614607e-03

102    1.078859e-22 -5.958530e-25     9.301358e-17       9.301809e-17

103   -4.786400e-16  3.004345e-17     2.733352e-01       2.733352e-01


     Disorder_incident  Fraud_incident  ...         PT998
PT999  \
0         1.915104e-04    1.102238e-05  ...  1.082971e-05  3.330119e-
06
1        -1.050827e-04   -2.566420e-06  ...  6.669576e-06 -2.072053e-
05
2         8.728162e-04    4.787485e-05  ...  2.157065e-05  2.422335e-
06
3         1.593893e-02    3.523578e-05  ...  4.881507e-04  6.932930e-
04
4         7.112700e-02    1.643208e-03  ...  6.925645e-03  4.925388e-
03
..                 ...             ...  ...           ...           ..
.
99        6.144110e-03    6.144110e-03  ... -3.346040e-02 -3.346040e-
02
100      -4.669138e-03   -4.669138e-03  ...  4.006013e-02  4.006013e-
02
101       1.614607e-03    1.614607e-03  ...  3.554252e-04  3.554252e-
04
102       9.302078e-17    9.299914e-17  ...  7.320177e-18  7.328255e-
18
103       2.733352e-01    2.733352e-01  ... -5.798732e-02 -5.798732e-
02

            RT061         RT199         RT442         RT226
RT234  \
0    4.649741e-06  3.529127e-05  0.000000e+00  1.559284e-06
6.142906e-06
1   -2.373240e-06  7.970685e-06 -0.000000e+00  6.888948e-06 -
5.693704e-06
2   -6.854610e-06 -1.825226e-05 -0.000000e+00  5.213532e-05 -
3.766178e-05
3    2.156524e-05  1.553076e-03  0.000000e+00  2.565115e-04
1.565075e-03
4    1.769189e-03  2.132296e-02 -0.000000e+00  3.535655e-03
8.762456e-03
..            ...           ...           ...           ...
...
```

```
99  -3.346040e-02 -3.346040e-02 -6.994405e-15 -3.346040e-02 -
3.346040e-02
100  4.006013e-02  4.006013e-02 -7.216450e-15  4.006013e-02
4.006013e-02
101  3.554252e-04  3.554252e-04 -5.528911e-14  3.554252e-04
3.554252e-04
102  6.959851e-18  7.354228e-18  1.000000e+00  7.335430e-18
7.326922e-18
103 -5.798732e-02 -5.798732e-02  0.000000e+00 -5.798732e-02 -
5.798732e-02

            RT303           SEV  Licensing_all
0    2.845246e-06  6.628911e-07   8.237905e-04
1    1.148509e-06  6.926393e-06  -2.084315e-04
2    1.981483e-06  4.256116e-05   2.751693e-03
3   -7.225220e-05  1.926287e-04   9.707932e-02
4   -5.059513e-04  4.324749e-03   8.690140e-01
..            ...           ...            ...
99  -3.346040e-02 -3.346040e-02   3.346040e-02
100  4.006013e-02  4.006013e-02  -4.006013e-02
101  3.554252e-04  3.554252e-04  -3.554252e-04
102  7.392788e-18  7.333139e-18  -7.316710e-18
103 -5.798732e-02 -5.798732e-02   5.798732e-02

[104 rows x 104 columns]
```

```
print(df_PrincipleComp1)
```

```
     Total households   Mean Income  Median Income   Mode Income  \
0            0.000009  1.315619e-03   1.121764e-03  3.688047e-03
1           -0.000564 -3.810806e-01  -3.593581e-01 -8.184402e-01
2            0.001236  5.638265e-01   5.044154e-01 -5.742712e-01
3            0.016656  6.422260e-01  -2.560167e-01  1.877961e-02
4            0.002744 -7.076987e-02   1.157051e-02 -1.883069e-04
..                ...           ...            ...           ...
99           0.000000 -5.689351e-17   1.547255e-16  5.827451e-18
100          0.000000  7.803545e-17  -1.794623e-16 -4.897021e-18
101          0.000000 -1.384899e-18   3.116636e-18 -1.718613e-19
102          0.000000  2.611670e-23  -1.044111e-22  3.137987e-24
103          0.000000 -1.972028e-16   5.290267e-16 -2.593525e-18

     Lower Quartile      LonAmALL  Damage_incident  Burglary_incident
\
0      6.827289e-04  9.999909e-01     2.915862e-05       2.612658e-05

1     -2.361731e-01  4.084497e-03    -2.113598e-05      -7.861451e-05

2      3.128134e-01  5.921352e-04     1.663442e-04       2.620105e-04

3     -7.119807e-01 -2.631768e-04     2.790686e-03       3.940450e-03
```

```
4      9.617322e-02 -9.757615e-04     1.167085e-02       1.786737e-02

..              ...           ...               ...                ...

99    -1.356456e-16  6.375711e-19     6.144110e-03       6.144110e-03

100    1.511652e-16  3.672929e-18    -4.669138e-03      -4.669138e-03

101   -3.066249e-18  1.826966e-19     1.614607e-03       1.614607e-03

102    1.078859e-22 -5.958530e-25     9.301358e-17       9.301809e-17

103   -4.786400e-16  3.004345e-17     2.733352e-01       2.733352e-01


     Disorder_incident  Fraud_incident  ...         PT998
PT999  \
0         1.915104e-04    1.102238e-05  ...  1.082971e-05  3.330119e-
06
1        -1.050827e-04   -2.566420e-06  ...  6.669576e-06 -2.072053e-
05
2         8.728162e-04    4.787485e-05  ...  2.157065e-05  2.422335e-
06
3         1.593893e-02    3.523578e-05  ...  4.881507e-04  6.932930e-
04
4         7.112700e-02    1.643208e-03  ...  6.925645e-03  4.925388e-
03
..                 ...             ...  ...           ...           ..
.
99        6.144110e-03    6.144110e-03  ... -3.346040e-02 -3.346040e-
02
100      -4.669138e-03   -4.669138e-03  ...  4.006013e-02  4.006013e-
02
101       1.614607e-03    1.614607e-03  ...  3.554252e-04  3.554252e-
04
102       9.302078e-17    9.299914e-17  ...  7.320177e-18  7.328255e-
18
103       2.733352e-01    2.733352e-01  ... -5.798732e-02 -5.798732e-
02

            RT061         RT199         RT442         RT226
RT234  \
0    4.649741e-06  3.529127e-05  0.000000e+00  1.559284e-06
6.142906e-06
1   -2.373240e-06  7.970685e-06 -0.000000e+00  6.888948e-06 -
5.693704e-06
2   -6.854610e-06 -1.825226e-05 -0.000000e+00  5.213532e-05 -
3.766178e-05
```

```
3    2.156524e-05  1.553076e-03  0.000000e+00  2.565115e-04
1.565075e-03
4    1.769189e-03  2.132296e-02 -0.000000e+00  3.535655e-03
8.762456e-03
..            ...           ...           ...           ...
...
99  -3.346040e-02 -3.346040e-02 -6.994405e-15 -3.346040e-02 -
3.346040e-02
100  4.006013e-02  4.006013e-02 -7.216450e-15  4.006013e-02
4.006013e-02
101  3.554252e-04  3.554252e-04 -5.528911e-14  3.554252e-04
3.554252e-04
102  6.959851e-18  7.354228e-18  1.000000e+00  7.335430e-18
7.326922e-18
103 -5.798732e-02 -5.798732e-02  0.000000e+00 -5.798732e-02 -
5.798732e-02

            RT303           SEV  Licensing_all
0    2.845246e-06  6.628911e-07   8.237905e-04
1    1.148509e-06  6.926393e-06  -2.084315e-04
2    1.981483e-06  4.256116e-05   2.751693e-03
3   -7.225220e-05  1.926287e-04   9.707932e-02
4   -5.059513e-04  4.324749e-03   8.690140e-01
..            ...           ...            ...
99  -3.346040e-02 -3.346040e-02   3.346040e-02
100  4.006013e-02  4.006013e-02  -4.006013e-02
101  3.554252e-04  3.554252e-04  -3.554252e-04
102  7.392788e-18  7.333139e-18  -7.316710e-18
103 -5.798732e-02 -5.798732e-02   5.798732e-02

[104 rows x 104 columns]
```

```
df_PC1 = pd.DataFrame(df_PrincipleComp1)
```

```
df_PC1.sample(10)
```

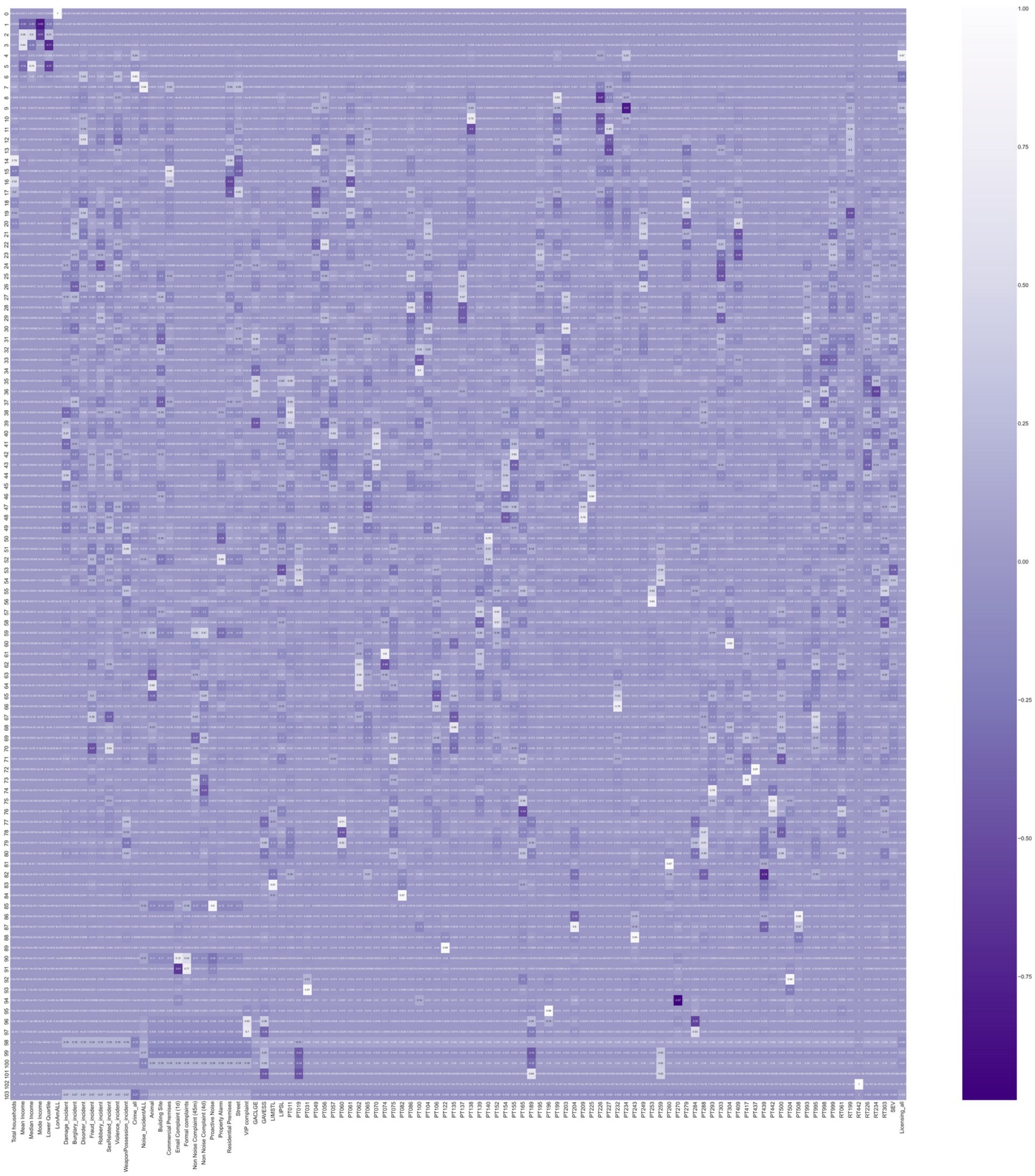
{"version_major":2,"version_minor":0,"model_id":"7ecbcd21cdc24c2e88f06
e5ca9256a00"}

{"version_major":2,"version_minor":0,"model_id":"048881a1994e49e3abc08
e4fad25b5f0"}

```
df_PC1.info()
```

```
<class 'lux.core.frame.LuxDataFrame'>
RangeIndex: 104 entries, 0 to 103
Columns: 104 entries, Total households to Licensing_all
dtypes: float64(104)
memory usage: 84.6 KB
```

```
sns.set(font_scale=5)
plt.figure(figsize=(200,200))
sns.heatmap(df_PrincipleComp1, annot=True, annot_kws={"size": 25},
cmap='Purples_r')
plt.show()
```



Plotting Each Component vs. Original Data

The next step is to visualize the data resulting from each component rather than the explained variance. We will use the inverse_transform method of the PCA model, and this will take each of the components and transform them back into the original data scale.

```
sns.set(font_scale=1)
```

```python
def transform_pca(X1, n):

    pca = PCA(n_components=n)
    pca.fit(X1)
    X1_new = pca.inverse_transform(pca.transform(X1))

    return X1_new

rows = 10
cols = 11
comps = 1

scaler = StandardScaler()
X1_scaled = scaler.fit_transform(X1)

fig, axes = plt.subplots(rows,
                         cols,
                         figsize=(25,25),
                         sharex=True,
                         sharey=True)


for row in range(rows):
    for col in range(cols):
        try:
            X1_new = transform_pca(X1_scaled, comps)
            ax = sns.scatterplot(x=X1_scaled[:, 0],
                                 y=X1_scaled[:, 1],
                                 ax=axes[row, col],
                                 color='red',
                                 alpha=.3)
            ax = sns.scatterplot(x=X1_new[:, 0],
                                 y=X1_new[:, 1],
                                 ax=axes[row, col],
                                 color='black')
            ax.set_title(f'PCA Components: {comps}');

            comps += 1
        except:
            pass
plt.tight_layout()
#plt.savefig('pcavisualize_2.png', dpi=300)
```


```
print('Explained variance ratio of each component:')
print(pca1.explained_variance_ratio_)
```

```
Explained variance ratio of each component:
[9.95314245e-01 4.37592002e-03 3.03911800e-04 4.09515758e-06
 1.36778698e-06 1.94668733e-07 1.24216435e-07 2.47410545e-08
 2.26834201e-08 1.92320069e-08 1.18310246e-08 8.71849128e-09
 7.98793664e-09 6.91972335e-09 5.79568672e-09 5.04210293e-09
 4.32490597e-09 3.82036112e-09 2.90555356e-09 2.81277089e-09
 2.52498370e-09 1.50764294e-09 1.47561759e-09 1.06518931e-09
 9.98683692e-10 7.45400769e-10 6.93264609e-10 6.06460800e-10
 4.92588192e-10 4.24161126e-10 3.77089548e-10 3.33624291e-10
 2.67637659e-10 2.39840010e-10 2.15506938e-10 1.98480251e-10
 1.96631217e-10 1.94446608e-10 1.67733942e-10 1.47010060e-10
 1.24836839e-10 1.08053210e-10 1.02748758e-10 1.02535165e-10
 8.87709677e-11 8.21637684e-11 7.76214759e-11 6.87454057e-11
 6.62897587e-11 5.47486573e-11 4.56938992e-11 4.22225251e-11
```

```
 3.97874952e-11 3.30252001e-11 3.12074001e-11 2.18313381e-11
 2.08832655e-11 1.67821728e-11 1.58793896e-11 1.52333884e-11
 1.41468807e-11 1.21886920e-11 1.13180762e-11 1.07485746e-11
 1.01516645e-11 9.65229694e-12 8.84747717e-12 8.47618045e-12
 7.99040924e-12 7.53724138e-12 6.78041994e-12 6.67329187e-12
 6.22243667e-12 5.59372891e-12 4.13158623e-12 3.45502960e-12
 2.93376398e-12 2.31780369e-12 2.17124400e-12 1.99224906e-12
 1.45681327e-12 8.19890314e-13 7.42595481e-13 6.60821680e-13
 6.22554400e-13 3.35018214e-13 2.42029176e-13 2.26721364e-13
 1.92833878e-13 1.20762995e-13 8.27061583e-14 7.48127669e-14
 3.46184400e-14 2.73506692e-14 1.92099585e-14 3.31657556e-15
 2.12736196e-15 1.90035591e-15 9.09051985e-33 9.09051985e-33
 9.09051985e-33 9.09051985e-33 9.09051985e-33 9.09051985e-33]
```

each of the principal components is summed together, and the total of all components will equal 1

```
def get_variance(X1, n):
    scaler = StandardScaler()
    pca1 = PCA(n_components=n)

    pca1.fit(scaler.fit_transform(X1))

    return pca1.explained_variance_ratio_.cumsum()[-1:]
for i in range(1,104):
    print('Components:\t', i, '=\t', get_variance(X1, i),
          '\tCumulative Variance')
```

```
Components:      1 =  [0.17632797]   Cumulative Variance
Components:      2 =  [0.23438892]   Cumulative Variance
Components:      3 =  [0.27685359]   Cumulative Variance
Components:      4 =  [0.31640795]   Cumulative Variance
Components:      5 =  [0.35256406]   Cumulative Variance
Components:      6 =  [0.38624207]   Cumulative Variance
Components:      7 =  [0.41097939]   Cumulative Variance
Components:      8 =  [0.43401258]   Cumulative Variance
Components:      9 =  [0.45684652]   Cumulative Variance
Components:      10 = [0.4779365]    Cumulative Variance
Components:      11 = [0.49779338]   Cumulative Variance
Components:      12 = [0.51752636]   Cumulative Variance
Components:      13 = [0.53618161]   Cumulative Variance
Components:      14 = [0.55427902]   Cumulative Variance
Components:      15 = [0.57061894]   Cumulative Variance
Components:      16 = [0.58703456]   Cumulative Variance
Components:      17 = [0.60207624]   Cumulative Variance
Components:      18 = [0.6164299]    Cumulative Variance
Components:      19 = [0.63088091]   Cumulative Variance
Components:      20 = [0.64427485]   Cumulative Variance
Components:      21 = [0.65702496]   Cumulative Variance
Components:      22 = [0.66945774]   Cumulative Variance
Components:      23 = [0.68148817]   Cumulative Variance
```

```
Components:      24 =  [0.69292533]   Cumulative Variance
Components:      25 =  [0.70548198]   Cumulative Variance
Components:      26 =  [0.71636097]   Cumulative Variance
Components:      27 =  [0.72752162]   Cumulative Variance
Components:      28 =  [0.73891355]   Cumulative Variance
Components:      29 =  [0.74912882]   Cumulative Variance
Components:      30 =  [0.75955839]   Cumulative Variance
Components:      31 =  [0.77027369]   Cumulative Variance
Components:      32 =  [0.78024759]   Cumulative Variance
Components:      33 =  [0.7897386]    Cumulative Variance
Components:      34 =  [0.79988463]   Cumulative Variance
Components:      35 =  [0.8094349]    Cumulative Variance
Components:      36 =  [0.81908005]   Cumulative Variance
Components:      37 =  [0.82883708]   Cumulative Variance
Components:      38 =  [0.83808375]   Cumulative Variance
Components:      39 =  [0.84708265]   Cumulative Variance
Components:      40 =  [0.85587678]   Cumulative Variance
Components:      41 =  [0.86379379]   Cumulative Variance
Components:      42 =  [0.87241567]   Cumulative Variance
Components:      43 =  [0.88040853]   Cumulative Variance
Components:      44 =  [0.887707]     Cumulative Variance
Components:      45 =  [0.89513993]   Cumulative Variance
Components:      46 =  [0.902209]     Cumulative Variance
Components:      47 =  [0.90856734]   Cumulative Variance
Components:      48 =  [0.91454257]   Cumulative Variance
Components:      49 =  [0.92020912]   Cumulative Variance
Components:      50 =  [0.92560115]   Cumulative Variance
Components:      51 =  [0.93078834]   Cumulative Variance
Components:      52 =  [0.93573378]   Cumulative Variance
Components:      53 =  [0.94039665]   Cumulative Variance
Components:      54 =  [0.94481703]   Cumulative Variance
Components:      55 =  [0.94890528]   Cumulative Variance
Components:      56 =  [0.95268012]   Cumulative Variance
Components:      57 =  [0.95607772]   Cumulative Variance
Components:      58 =  [0.95946226]   Cumulative Variance
Components:      59 =  [0.96261269]   Cumulative Variance
Components:      60 =  [0.96567027]   Cumulative Variance
Components:      61 =  [0.96864328]   Cumulative Variance
Components:      62 =  [0.97118852]   Cumulative Variance
Components:      63 =  [0.97373982]   Cumulative Variance
Components:      64 =  [0.97618485]   Cumulative Variance
Components:      65 =  [0.97833739]   Cumulative Variance
Components:      66 =  [0.98028936]   Cumulative Variance
Components:      67 =  [0.98213471]   Cumulative Variance
Components:      68 =  [0.98386288]   Cumulative Variance
Components:      69 =  [0.98549134]   Cumulative Variance
Components:      70 =  [0.98701357]   Cumulative Variance
Components:      71 =  [0.98848794]   Cumulative Variance
Components:      72 =  [0.98989396]   Cumulative Variance
Components:      73 =  [0.99112651]   Cumulative Variance
```

```
Components:      74 = [0.99230892]   Cumulative Variance
Components:      75 = [0.99334772]   Cumulative Variance
Components:      76 = [0.99430184]   Cumulative Variance
Components:      77 = [0.99513909]   Cumulative Variance
Components:      78 = [0.99576534]   Cumulative Variance
Components:      79 = [0.99637905]   Cumulative Variance
Components:      80 = [0.99694114]   Cumulative Variance
Components:      81 = [0.99737672]   Cumulative Variance
Components:      82 = [0.99777641]   Cumulative Variance
Components:      83 = [0.99813161]   Cumulative Variance
Components:      84 = [0.9984621]    Cumulative Variance
Components:      85 = [0.99875125]   Cumulative Variance
Components:      86 = [0.99900723]   Cumulative Variance
Components:      87 = [0.9991999]    Cumulative Variance
Components:      88 = [0.99937768]   Cumulative Variance
Components:      89 = [0.99952631]   Cumulative Variance
Components:      90 = [0.9996476]    Cumulative Variance
Components:      91 = [0.99973583]   Cumulative Variance
Components:      92 = [0.99981722]   Cumulative Variance
Components:      93 = [0.99988381]   Cumulative Variance
Components:      94 = [0.9999467]    Cumulative Variance
Components:      95 = [0.99997202]   Cumulative Variance
Components:      96 = [0.99999485]   Cumulative Variance
Components:      97 = [0.99999792]   Cumulative Variance
Components:      98 = [1.]      Cumulative Variance
Components:      99 = [1.]      Cumulative Variance
Components:      100 =      [1.]      Cumulative Variance
Components:      101 =      [1.]      Cumulative Variance
Components:      102 =      [1.]      Cumulative Variance
Components:      103 =      [1.]      Cumulative Variance
```

```
print('Eigenvalues of each component:')
print(pca1.explained_variance_)
```

```
Eigenvalues of each component:
[6.75780345e+11 2.97108251e+09 2.06344501e+08 2.78045554e+06
 9.28675101e+05 1.32172632e+05 8.43382137e+04 1.67982308e+04
 1.54011756e+04 1.30577979e+04 8.03281369e+03 5.91952248e+03
 5.42350379e+03 4.69822778e+03 3.93504985e+03 3.42339525e+03
 2.93644591e+03 2.59387924e+03 1.97275985e+03 1.90976397e+03
 1.71436746e+03 1.02363196e+03 1.00188796e+03 7.23222847e+02
 6.78068071e+02 5.06098643e+02 4.70700182e+02 4.11763713e+02
 3.34448562e+02 2.87989199e+02 2.56029396e+02 2.26518147e+02
 1.81715745e+02 1.62842204e+02 1.46320977e+02 1.34760507e+02
 1.33505084e+02 1.32021818e+02 1.13884938e+02 9.98142141e+01
 8.47594438e+01 7.33640016e+01 6.97624808e+01 6.96174597e+01
 6.02720955e+01 5.57860596e+01 5.27020165e+01 4.66755040e+01
 4.50082135e+01 3.71722466e+01 3.10244117e+01 2.86674814e+01
 2.70141891e+01 2.24228491e+01 2.11886323e+01 1.48226445e+01
 1.41789394e+01 1.13944542e+01 1.07814988e+01 1.03428887e+01
 9.60519149e+00 8.27565617e+00 7.68454131e+00 7.29787152e+00
```

```
 6.89259231e+00 6.55354084e+00 6.00709896e+00 5.75500268e+00
 5.42518259e+00 5.11749892e+00 4.60364609e+00 4.53091022e+00
 4.22479676e+00 3.79792821e+00 2.80518919e+00 2.34583310e+00
 1.99191365e+00 1.57370015e+00 1.47419172e+00 1.35266100e+00
 9.89120550e-01 5.56674198e-01 5.04193960e-01 4.48672673e-01
 4.22690652e-01 2.27464567e-01 1.64328563e-01 1.53935144e-01
 1.30926836e-01 8.19934595e-02 5.61543216e-02 5.07950104e-02
 2.35045981e-02 1.85700594e-02 1.30428279e-02 2.25182809e-03
 1.44439749e-03 1.29026906e-03 6.17211566e-21 6.17211566e-21
 6.17211566e-21 6.17211566e-21 6.17211566e-21 6.17211566e-21]
```

```
plt.figure(figsize=(8,8))
plt.plot(pca1.explained_variance_)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```

```
df = pd.DataFrame({'eigenvalue':pca1.explained_variance_,
             'PC':list(range(1, pca1.n_components_ + 1))})
df.plot.line(x = 'PC', y = 'eigenvalue')
```

Grouped dataset

```
# drop 'OAs' to create X - the main geographical element
X2 = CIA_Grid.drop(['CIA_Composite'], axis=1).values

# the name list of independent variables
list_var_X2 = CIA_Grid.columns.tolist()
list_var_X2.remove('CIA_Composite')


y2 = CIA_Grid.loc[:,['CIA_Composite']].values
```

```
from sklearn.preprocessing import StandardScaler
X2_std = StandardScaler().fit_transform(X2)
```

```
print(X2_std)
```

```
[[-0.24533923 -0.44559962 -0.16242058 -2.48691443 -1.97183075
2.12747807]
 [-0.24533923 -0.44559962 -0.16242058 -2.48691443 -1.97183075 -
0.71657354]
 [-0.24533923 -0.44559962 -0.16242058 -2.48691443 -1.97183075 -
0.66917268]
 ...
 [-0.24533923  0.7004626  -0.13191791 -1.01164056 -0.0453897  -
0.02587529]
 [-0.24533923 -0.1937992   0.38505728 -1.29674207  0.15118592
0.15018504]
 [-0.24533923 -0.43819373 -0.13320813 -0.44924061 -0.21575523 -
0.53374165]]
```

```
X2.shape
```

```
(850, 6)
```

```
df1 = pd.DataFrame(X2)
```

```
df1.sample(10)
```

{"version_major":2,"version_minor":0,"model_id":"210888922cf44f17a6c85eabb1655ad9"}

{"version_major":2,"version_minor":0,"model_id":"5a4b24afc9f04b68a3bb617e11463318"}

```
from sklearn.decomposition import PCA
rand_st_int = 10
pca2 = PCA(random_state=rand_st_int)
# fit the components
X2_new_components2 = pca2.fit_transform(X2)
```

```
print(X2_new_components2)
```

```
[[-1.33512360e+05  5.53686205e+04 -8.17842780e+01 -4.06126738e+01
   3.49006978e+02 -1.16679903e+02]
 [-1.33512390e+05  5.53688199e+04 -1.15007934e+02 -8.58001492e+01
  -6.08709772e+01 -4.41933388e+01]
 [-1.33512389e+05  5.53688166e+04 -1.14454206e+02 -8.50470246e+01
  -5.40396779e+01 -4.54014482e+01]
 ...
 [-1.08408948e+05  2.24517777e+04 -9.73708102e+00  4.83453855e+02
  -3.46314034e+01  1.71391635e+01]
 [ 3.16311911e+05  2.93769116e+04 -5.29269197e+02 -6.74425006e+01
   7.38966900e+01  5.31463108e+01]
 [-1.09452805e+05  9.89054375e+03 -2.10492686e+02 -1.20538511e+02
  -3.51360126e+01  1.44529646e+01]]
```

```
X2_new_components2.shape
```

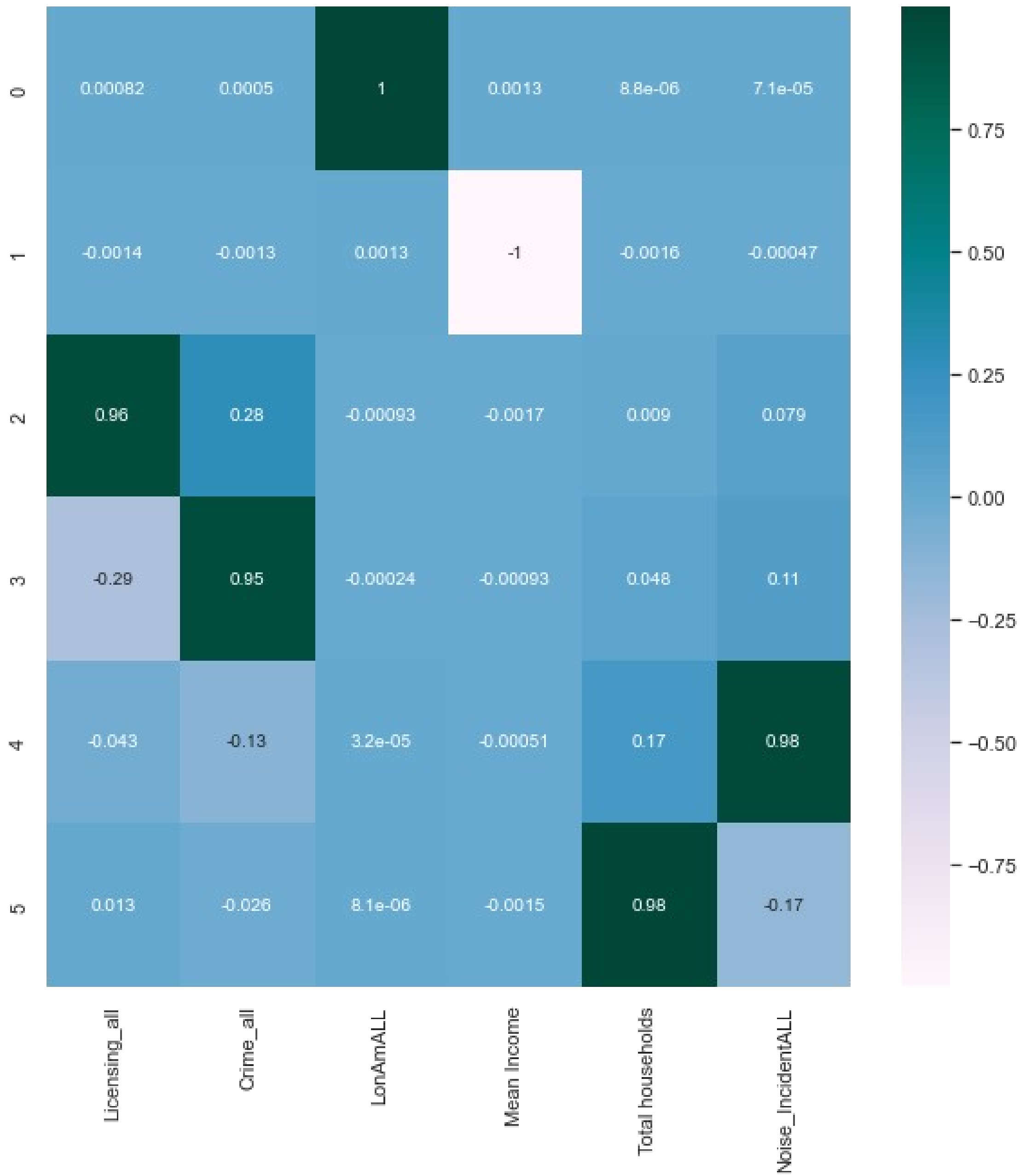
```
(850, 6)
```

```
df_PrincipleComp2 = pd.DataFrame(pca2.components_, columns =
list_var_X2)
df_PrincipleComp2
```

{"version_major":2,"version_minor":0,"model_id":"35d0579479aa4e40b5d33862a821c3cf"}

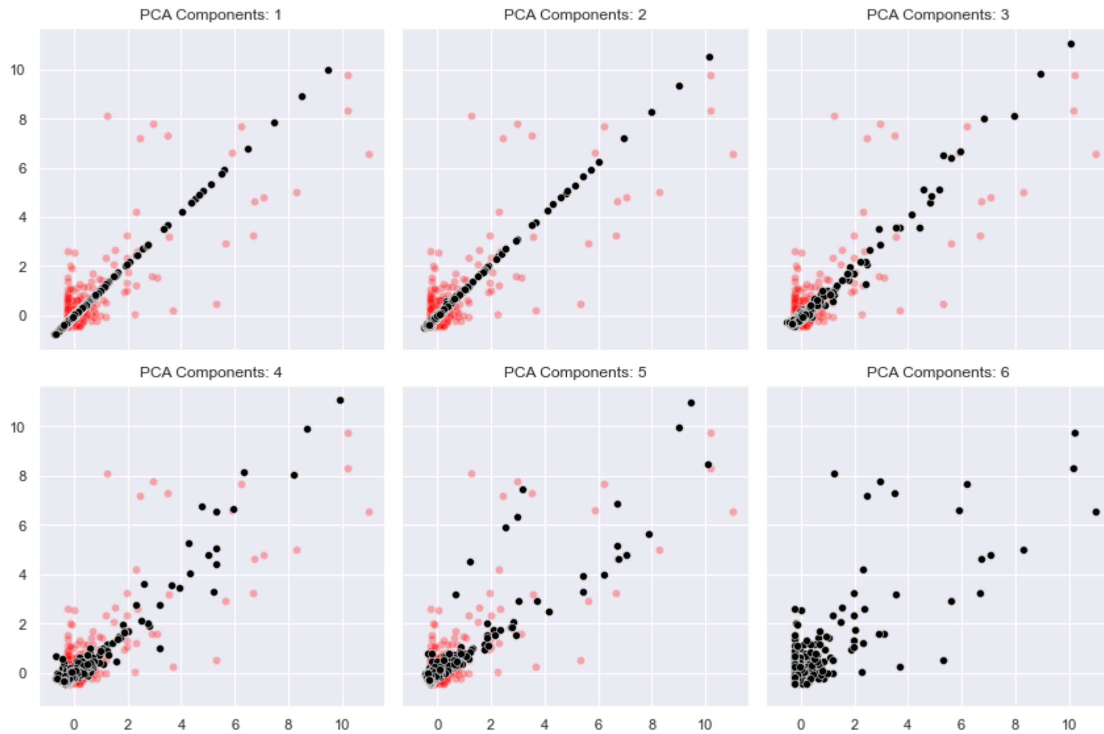
{"version_major":2,"version_minor":0,"model_id":"0616db1a8d4d48c79dab38070dbaa82e"}

```
plt.figure(figsize=(10,10))
sns.heatmap(df_PrincipleComp2, annot=True, cmap='PuBuGn')
plt.show()
```




```python
sns.set(font_scale=1)

def transform_pca(X2, n):

    pca = PCA(n_components=n)
    pca.fit(X2)
    X2_new = pca.inverse_transform(pca.transform(X2))

    return X2_new

rows = 2
cols = 3
```

```python
comps = 1

scaler = StandardScaler()
X2_scaled = scaler.fit_transform(X2)

fig, axes = plt.subplots(rows,
                         cols,
                         figsize=(12,8),
                         sharex=True,
                         sharey=True)


for row in range(rows):
    for col in range(cols):
        try:
            X2_new = transform_pca(X2_scaled, comps)
            ax = sns.scatterplot(x=X2_scaled[:, 0],
                                 y=X2_scaled[:, 1],
                                 ax=axes[row, col],
                                 color='red',
                                 alpha=.3)
            ax = sns.scatterplot(x=X2_new[:, 0],
                                 y=X2_new[:, 1],
                                 ax=axes[row, col],
                                 color='black')
            ax.set_title(f'PCA Components: {comps}');

            comps += 1
        except:
            pass
plt.tight_layout()
#plt.savefig('pcavisualize_2.png', dpi=300)
```




```
print('Explained variance ratio of each component:')
print(pca2.explained_variance_ratio_)
```

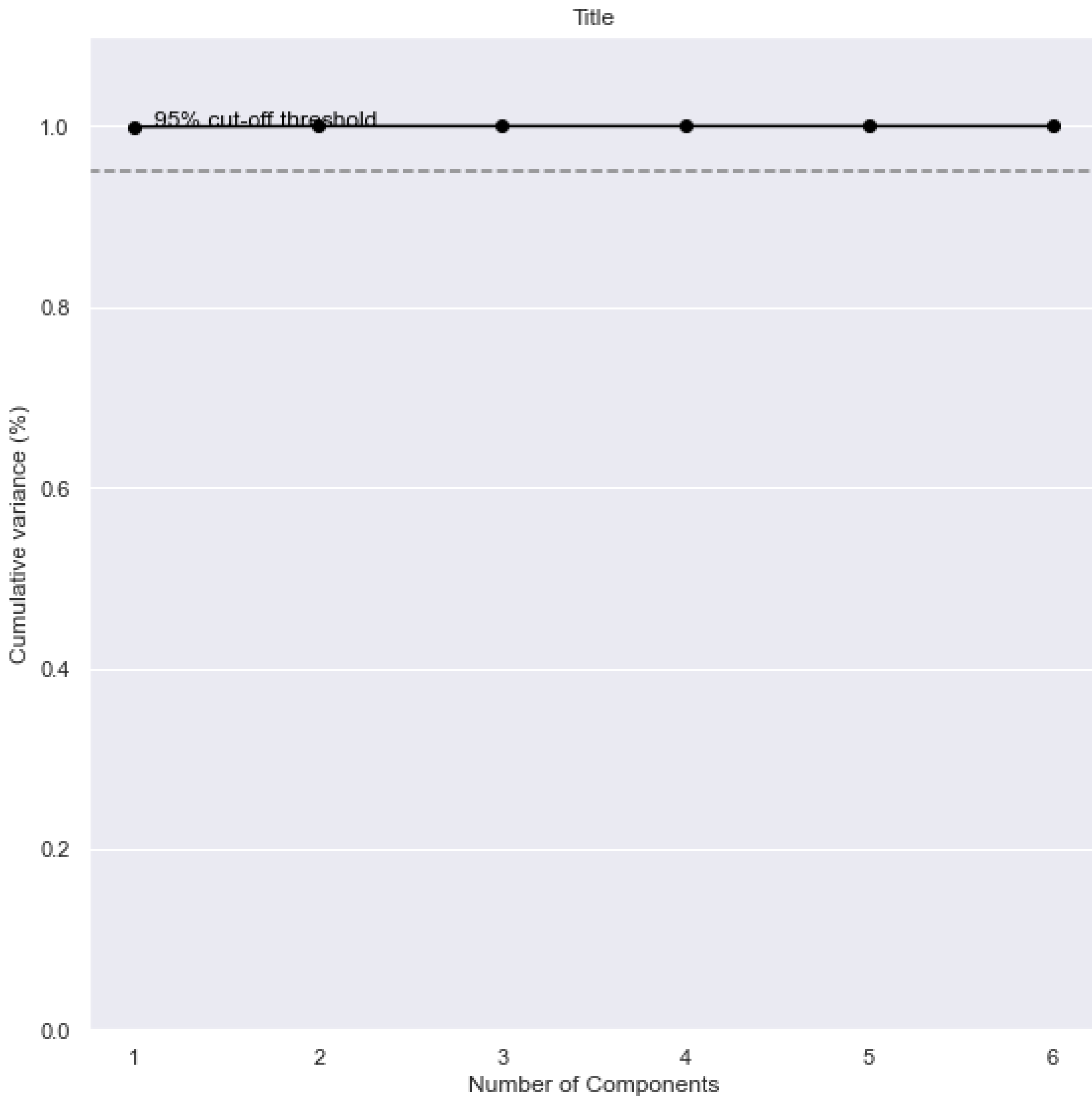
```
Explained variance ratio of each component:
[9.99261939e-01 7.36757223e-04 1.18088316e-06 9.79386359e-08
 1.93721103e-08 5.89125590e-09]
```

```
def get_variance(X2, n):
    scaler = StandardScaler()
    pca2 = PCA(n_components=n)

    pca2.fit(scaler.fit_transform(X2))

    return pca2.explained_variance_ratio_.cumsum()[-1:]
for i in range(1,7):
    print('Components:\t', i, '=\t', get_variance(X2, i),
          '\tCumulative Variance')
```

```
Components:      1 =   [0.49911426]    Cumulative Variance
Components:      2 =   [0.73576719]    Cumulative Variance
Components:      3 =   [0.8445498]     Cumulative Variance
Components:      4 =   [0.92273902]    Cumulative Variance
Components:      5 =   [0.97437188]    Cumulative Variance
Components:      6 =   [1.]      Cumulative Variance
```

```
scaler = StandardScaler()
data_rescaled = scaler.fit_transform(X2)
```

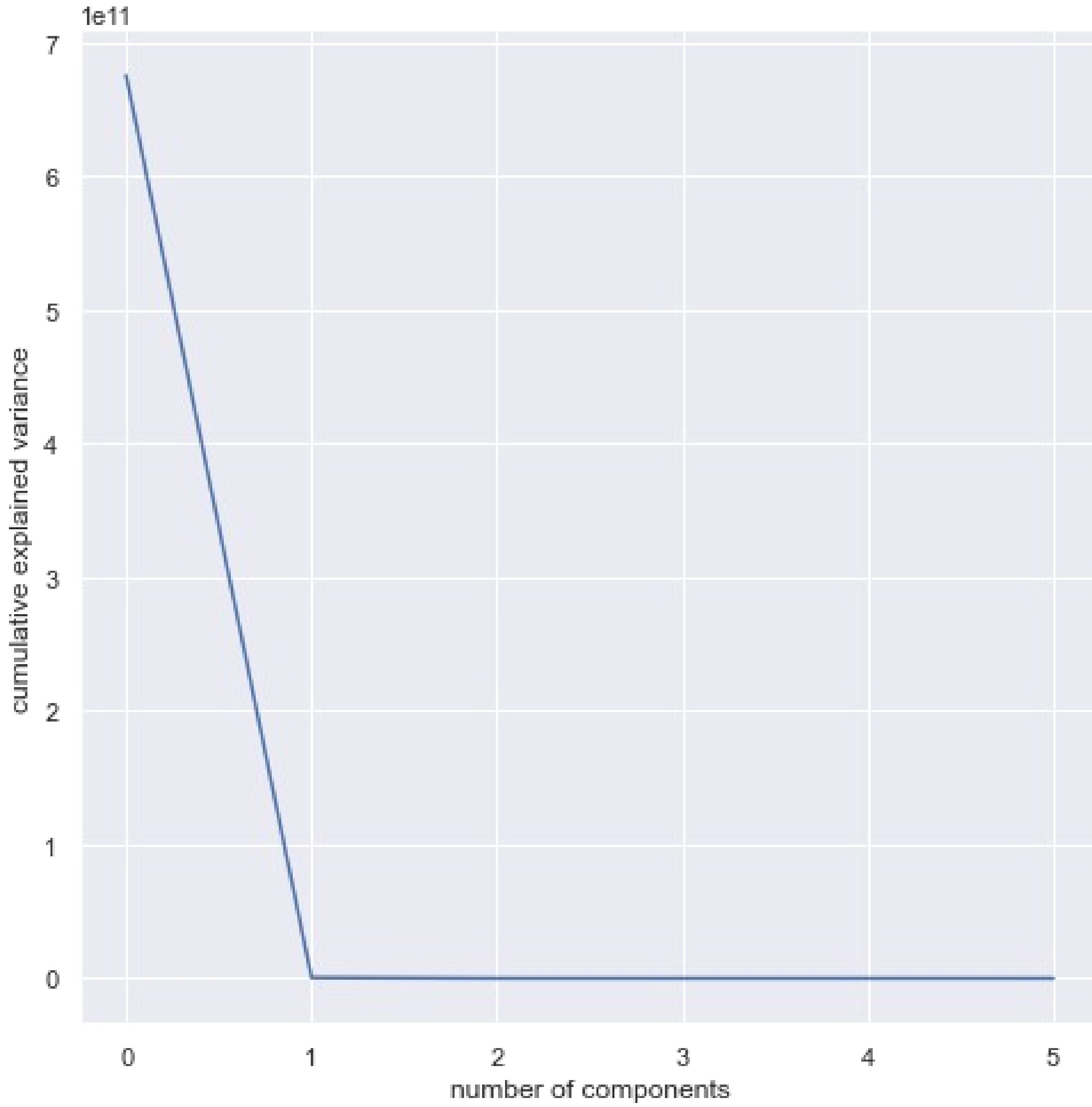
```
pca3 = PCA().fit(data_rescaled)

plt.rcParams["figure.figsize"] = (8,8)

fig, ax = plt.subplots()
xi = np.arange(1, 7, step=1)
y = np.cumsum(pca2.explained_variance_ratio_)

plt.ylim(0.0,1.1)
plt.plot(xi, y, marker='o', linestyle='-', color='black')

plt.xlabel('Number of Components')
plt.xticks(np.arange(1, 7, step=1))
plt.ylabel('Cumulative variance (%)')
plt.title('Title')

plt.axhline(y=0.95, color='grey', linestyle='--')
plt.text(1.1, 1, '95% cut-off threshold', color = 'black',
fontsize=12)

ax.grid(axis='x')
plt.tight_layout()
#plt.savefig('pcavisualize_1.png', dpi=300)
plt.show()
```




```
print('Eigenvalues of each component:')
print(pca2_explained_variance)
```

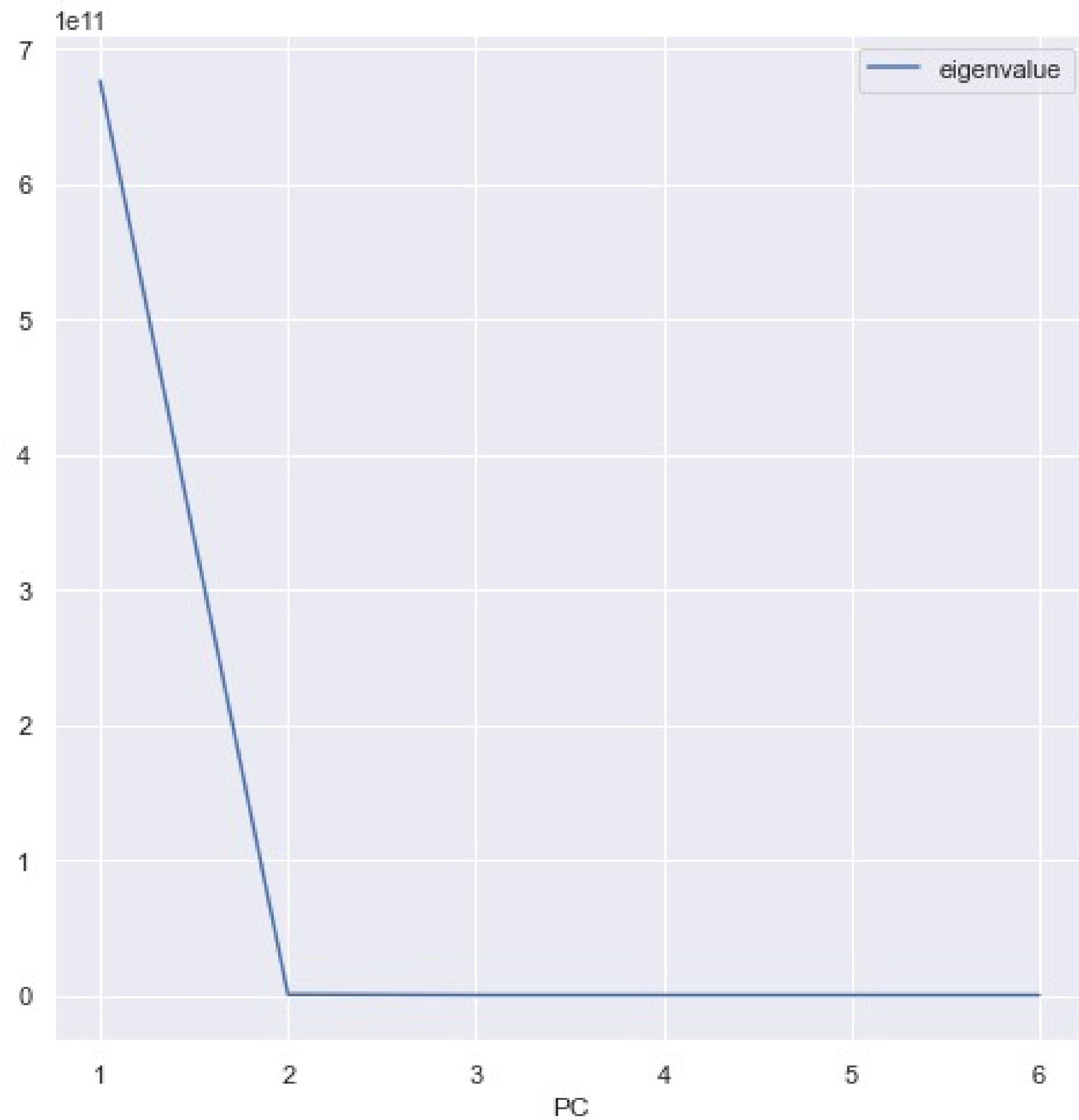
```
Eigenvalues of each component:
[6.75769916e+11 4.98246103e+08 7.98594725e+05 6.62328677e+04
 1.31007585e+04 3.98407399e+03]
```

```
plt.figure(figsize=(8,8))
plt.plot(pca2_explained_variance)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()

#PCA1 is at 0 in xscale
```




```
df = pd.DataFrame({'eigenvalue':pca2.explained_variance_,
             'PC':list(range(1, pca2.n_components_ + 1))})
df.plot.line(x = 'PC', y = 'eigenvalue')
```

```
<AxesSubplot:xlabel='PC'>
```


1e11
7
6
5
4
3
2
1
0
eigenvalue
1
2
3
4
5
6
PC