School of Engineering and Applied Science (SEAS), Ahmedabad University

B.Tech (CSE Semester VI):
Machine Learning (CSE 523)

Project Submission 3: Principal Component Analysis

Submission Deadline: April 05, 2020 (11:59 PM)

- **Group No.:** 21
- **Project Area:** Environment and Climate Change
- **Project Title:** Air Quality Index prediction using machine learning algorithms
- **Name of the group members :**
  1. Vishal Saha (AU1741004)
  2. Rushil Shah (AU1741009)
  3. Muskan Matwani (AU1741027)
  4. Mohit Vaswani (AU1741039)

Submission Guidelines

1. The objective of this project submission is to apply the Principal Component Analysis (PCA) to your domain problem and derive various inferences. Consider the following two cases:

   Case 1: Considering that the dimensionality of your dataset is smaller than the number of samples.

   1. **Dataset Description** : Our dataset consists of the following features - stncode, sampling date, state, location, agency, type, so2, no2, rspm, spm, locationmonitoringstation, pm25, date. It consists of 435742 rows and 13 columns and one output label - AQI which is constructed based on the concentration of pollutants in the dataset. Out of 13 features, only 4 features namely pollutant concentrations of SO_2, NO_2, RSPM and SPM are useful in predicting the output label - AQI. The feature matrix would be of the dimension - (435742, 4). Since the number of samples are greater than the number of features, hence we can apply case 1 as explained from below steps.

   2. **Data Preprocessing, Feature correlaton analysis and formation of input matrix X** : We at first removed the unnecessary columns from the dataset such as - stncode, agency, samplingdate, location, monitoring station. Further, we removed the outliers from each of the pollutant columns as outliers have a major effect on the mean of the feature data as we do not know the distribution of data. Further, Null values in the pollutant's feature column were replaced by the mean of the column. Finally, we Calculate AQI and pollutant index of each of the pollutants by using the EPA method as described below:

The pollutants/Independent variables are: NO_2, SO_2, RSPM (Respirable suspended particulate matter), and SPM (Suspended particulate matter).
The target/dependent variable : AQI (Air quality index)

Given all the pollutants concentration in the dataset, we find pollutant index by the following formula:

$$AQI_P = AQI_{min} + \frac{P_{Obs} - P_{Min}}{(P_{Max} - P_{Min})}(AQI_{Max} - AQI_{Min})$$

where, P_{Obs} = observed 24-hour average concentration in µg/m3
P_{Max} = maximum concentration of AQI color category that contains P_{Obs}
P_{Min} = minimum concentration of AQI color category that contains P_{Obs}
AQI_{Max} = maximum AQI value for color category that corresponds to P_{Obs}
AQI_{Min} = minimum AQI value for color category that corresponds to P_{Obs}

We calculate pollutant indexes (AQI_P) of each of the pollutants namely - si, ni, rpi and spi and append it in the dataset. Then, we find AQI by taking maximum of all the pollutant indexes of the pollutants :

$$AQI = max(AQI_{NO2}, AQI_{SO2}, AQI_{RSPM}, AQI_{SPM})$$

We have got one column 'AQI' in our dataset. Further analysis would be done on the same.

X is formed by taking the four pollutant AQI indexes namely - si, ni, rpi and spi columns from the dataset. Dimension of X is - 435742 X 4
y is formed by taking AQI columns from the dataset. Dimension of y is - 435742 X 1

Correlation among features:
Next, we plotted the correlation matrix whose dimension is 4X4 (taking 4 features) that depicts pairwise correlation of columns. Correlation shows how the two variables are related to each other. Positive values shows as one variable increases other variable increases as well. Negative values shows as one variable increases other variable decreases.Bigger the values,more strongly two variables are correlated and vice-versa.

3. **Normalization, Formation of Co-variance matrix and eigenvalues and vectors calculation**:
After computing feature matrix X_n, we normalize the values by subtracting mean of X_n from it and then further dividing it by the standard deviation:

$$\mathbf{X} = (\mathbf{X_n} - \bar{}) / \boldsymbol{\text{œ}}$$

where μ is the mean of data points in X and σ is the standard deviation of X

Then computing the Covariance matrix S, using the following equation:

$$S = \frac{1}{N}(\mathbf{X^T X})$$

where N is the total number of data points.
The dimension of S is = 4 X 4

We compute eigenvalues and eigenvectors of the covariance matrix S using python library function and sort the eigenvalues in descending order and sorting the eigenvectors corresponding to the eigenvalues order.

Since, from the previous analysis, we know that

$$V_m = \lambda_m$$

This means that the variance of the data, when projected onto an M-dimensional subspace, equals the sum of the eigenvalues that are associated with the corresponding eigen vectors of the data co-variance matrix.

hence from this, we calculate the explained variance (EV) that tells us how much information (variance) can be attributed to each of the features present from the following formula :

$$EV_i = (\lambda_i / total) * 100$$

where total is the sum of the eigenvalues and EV_i is the percentage of variance attributed to the ith feature. This way we plot a graph that shows the importance of the features by depicting the contribution in terms of variance percentage made by each feature. By looking at the graph, we concluded that 1st principal component contributes to 50% and 2nd one contributes as much to 25% and the rest 2 features contributes less comparatively. Hence, we choose M = 2 (number of principal components taken for analysis).

4. **Finding projection matrix and constructing the projected data matrix**:

To find the projection matrix, we first assign a matrix B to the top eigenvectors corresponding to M greatest eigenvalues and apply the below formula:

$$P = B.B^T$$

where P = projection matrix
The dimension of projection matrix = (4, 4)

We know, $\bar{X}$ can be reconstructed by multiplying the projection matrix with X,

$$\bar{X} = BB^T X = PX$$

where $\bar{X}$ is projected data matrix (reconstructed)
The dimension of = (435742, 4)

5. **Finding the unnormalized projected data matrix and computing reconstructed AQI from this projected data matrix** :

We first compute the normalized version of the computed projected data matrix by multiplying it with standard deviation and adding the mean of X computed earlier like:

$$X_f = \bar{X} * \sigma + \mu$$

where X_f is the unnormalized projected data matrix,
σ is the standard deviation of feature matrix X,
μ is the mean of feature matrix X, and
$\bar{X}$ is the projected data matrix (normalized)

After this, we compute the AQI based on the input matrix which is the normalized projected data matrix and label that column as $AQI_{reconstructed}$. Then we compute the MSE error between the actual and reconstructed AQI to analyze the effect of the dimensionality reduction on the output labels of data as well.

AQI	AQI_reconstructed
166.840252	174.691802
166.840252	171.120923
166.840252	179.002764
166.840252	171.423756
166.840252	169.461780

The above graph shows 5 values of actual AQI and AQI reconstructed from the projected data matrix.

6. **Analysis for each principal components** :
Since we have maximum of 4 features, we take all values of M ranging from 1 to 4. Apply all the above steps and find the projected data matrix in each case. We find MSE error between the projected data matrix and X for each of the values of principal components. We then plot the loss (MSE) w.r.t each principal component.

Moreover, to find an M-dimensional subspace of RD that retains as much information as possible, PCA tells us to choose the columns of the matrix B in as the M eigenvectors of the data covariance matrix S that are associated with the M largest eigenvalues. The maximum amount of variance PCA can capture with the first M principal components is :

$$V_M = \sum_{m=1}^{M} \lambda_m$$

where λm is the mth eigenvalue of the covariance matrix S.

We then plot the variance captures w.r.t each principal component ranging from M =1 to 4.

7. **Analysis for unordered eigenvalues and its effect on MSE** :
As in PCA funcntion, we sort the eigenvalues in decreasing order and eigenvectors corresponding to the eigenvalues, hence for analysis, we see the effect of unsorted eigenvalues on MSE error. The graphs are attached in the inference section. As we dont sort the order of eigenvalues, then taking first principal components wont be the same taking top principal components that amounts to highest variance. Hence, there would be change in the MSE error but the decreasing nature of MSE will remain same. Since we have too few features, the effect of unordered eigenvalues is not much reflected in the graph.

Case 2: Considering that the dimensionality of your dataset is larger than the number of samples used.
Justification: Case 2 says that dataset size should be such that number of rows are lesser than the number of features/columns. However, in our case, since the dataset size is (435742, 4), its not possible to take fewer rows than columns. However, since the features are the pollutants are independent and contribute independently to find the AQI, its not possible to increase the dimension of the dataset by using the available features. Moreover, since the row size of the dataset is really huge (435742), hence taking even 10% of the dataset amounts to more than 60,000 rows and computing MSE on this dataset doesn't have much difference. The below picture depicts the percentage of the data retained and MSE error (after taking the projected data matrix and comparing with original feature matrix):

	0	1
0	10	0.376084
1	20	0.361571
2	30	0.383691
3	40	0.383702
4	50	0.381058
5	60	0.373661
6	70	0.376671
7	80	0.377851
8	90	0.379173

The first column shows the percent of the data taken out of total and the second column shows the MSE error calculated between projected data matrix and original feature matrix. Since, the dataset is too large, we see there is not much effect on MSE. So, **case 2 cannot be applied on our dataset**. However, if it would have been possible to apply PCA on a dataset having more features than rows, then it would have resulted into overfitting the dataset and it would do good on training but not very well on the unseen/testing data.

2. **URL links:** For PCA code and dataset, click here
3. **Implementation code:** From next page

Principal Component Analysis - Implmentation Code

April 5, 2020

[1]:
```
# Required Libraries
import numpy as np
import timeit
import matplotlib as mpl
mpl.use('Agg')
import matplotlib.pyplot as plt
plt.style.use('fivethirtyeight')
from ipywidgets import interact
from numpy import linalg
import pandas as pd
import seaborn as sns
import warnings; warnings.simplefilter('ignore')

%matplotlib inline
```

[7]:
```
dataset = pd.read_csv('data.csv',encoding= "ISO-8859-1") # dataset␣
 ↪incorporation
dataset.describe()
```

[7]:
```
                 so2            no2           rspm            spm        ␣
 ↪pm2_5
count  401096.000000  419509.000000  395520.000000  198355.000000  9314.
 ↪000000
mean       10.829414      25.809623     108.832784     220.783480    40.
 ↪791467
std        11.177187      18.503086      74.872430     151.395457    30.
 ↪832525
min         0.000000       0.000000       0.000000       0.000000     3.
 ↪000000
25%         5.000000      14.000000      56.000000     111.000000    24.
 ↪000000
50%         8.000000      22.000000      90.000000     187.000000    32.
 ↪000000
75%        13.700000      32.200000     142.000000     296.000000    46.
 ↪000000
max       909.000000     876.000000    6307.033333    3380.000000   504.
 ↪000000
```

[8]:
```
# DATA CLEANING AND PREPROCESSING STARTS HERE

# dropping the unnecessary columns from the dataset
dataset.
 ↪drop(['stn_code','agency','sampling_date','location_monitoring_station'],␣
 ↪axis=1,inplace=True)
dataset.info()# printing info
dataset.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 435742 entries, 0 to 435741
Data columns (total 9 columns):
state       435742 non-null object
location    435739 non-null object
type        430349 non-null object
so2         401096 non-null float64
no2         419509 non-null float64
rspm        395520 non-null float64
spm         198355 non-null float64
pm2_5       9314 non-null float64
date        435735 non-null object
dtypes: float64(5), object(4)
memory usage: 29.9+ MB
```

[8]:

	state	location	type	so2	no2
0	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	4.8	17.4
1	Andhra Pradesh	Hyderabad	Industrial Area	3.1	7.0
2	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.2	28.5
3	Andhra Pradesh	Hyderabad	Residential, Rural and other Areas	6.3	14.7
4	Andhra Pradesh	Hyderabad	Industrial Area	4.7	7.5

	rspm	spm	pm2_5	date
0	NaN	NaN	NaN	1990-02-01
1	NaN	NaN	NaN	1990-02-01
2	NaN	NaN	NaN	1990-02-01
3	NaN	NaN	NaN	1990-03-01
4	NaN	NaN	NaN	1990-03-01

[9]:
```
def remove_outlier(df_in, col_name):
    q1 = df_in[col_name].quantile(0.25) #it is middle number between␣
 ↪smallest value and median
    q3 = df_in[col_name].quantile(0.75) #it is middle number between␣
 ↪largest value and median
    iqr = q3-q1 #Interquartile range
    fence_low  = q1-1.5*iqr #any number suspected lower than this value is␣
 ↪suspected outlier
```

```
    fence_high = q3+1.5*iqr #any number suspected higher than this value is␣
 ↪suspected outlier
    df_out = df_in.loc[(df_in[col_name] > fence_low) & (df_in[col_name] <␣
 ↪fence_high)]  # limiting the dataset values to just the range where␣
 ↪outliers does not exists
    #return df_out
# calling function to remove outlier values from the pollutant columns of␣
 ↪the dataset
remove_outlier(dataset,'so2')
remove_outlier(dataset,'no2')
remove_outlier(dataset,'rspm')
remove_outlier(dataset,'spm')
```

[10]:
```
by_State=dataset.groupby('state')#grouping dataset by state

# filling the nan values with the mean of that column

def impute_mean(series):
    return series.fillna(series.mean())

dataset['rspm'] = by_State['rspm'].transform(impute_mean)    #transforming␣
 ↪null values in rspm column
dataset['so2'] = by_State['so2'].transform(impute_mean)      #transforming␣
 ↪null values in so2 column
dataset['no2'] = by_State['no2'].transform(impute_mean)      #transforming␣
 ↪null values in no2 column
dataset['spm'] = by_State['spm'].transform(impute_mean)      #transforming␣
 ↪null values in spm column
dataset['pm2_5'] = by_State['pm2_5'].transform(impute_mean)  #transforming␣
 ↪null values in pm25 column
```

[11]:
```
#Missing values being filled in columns
for col in dataset.columns.values:
    if dataset[col].isnull().sum() == 0:
        continue
    if col == 'date':           # filling mean values in data point date
        guess_values = dataset.groupby('state')['date'].apply(lambda x: x.
 ↪mode().max())
    elif col=='type':          #filling mean values in data point type
        guess_values = dataset.groupby('state')['type'].apply(lambda x: x.
 ↪mode().max())
    else:                      #filling mean values in data point location
        guess_values = dataset.groupby('state')['location'].apply(lambda x:␣
 ↪x.mode().max())
dataset.head()
```

[11]:
```
            state   location                                type  so2   no2␣
 ↪ \
0  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  4.8  17.4
1  Andhra Pradesh  Hyderabad                 Industrial Area  3.1   7.0
2  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  6.2  28.5
3  Andhra Pradesh  Hyderabad  Residential, Rural and other Areas  6.3  14.7
4  Andhra Pradesh  Hyderabad                 Industrial Area  4.7   7.5

        rspm         spm  pm2_5        date
0  78.182824  200.260378    NaN  1990-02-01
1  78.182824  200.260378    NaN  1990-02-01
2  78.182824  200.260378    NaN  1990-02-01
3  78.182824  200.260378    NaN  1990-03-01
4  78.182824  200.260378    NaN  1990-03-01
```

0.0.1 Correlation among original pollutant features

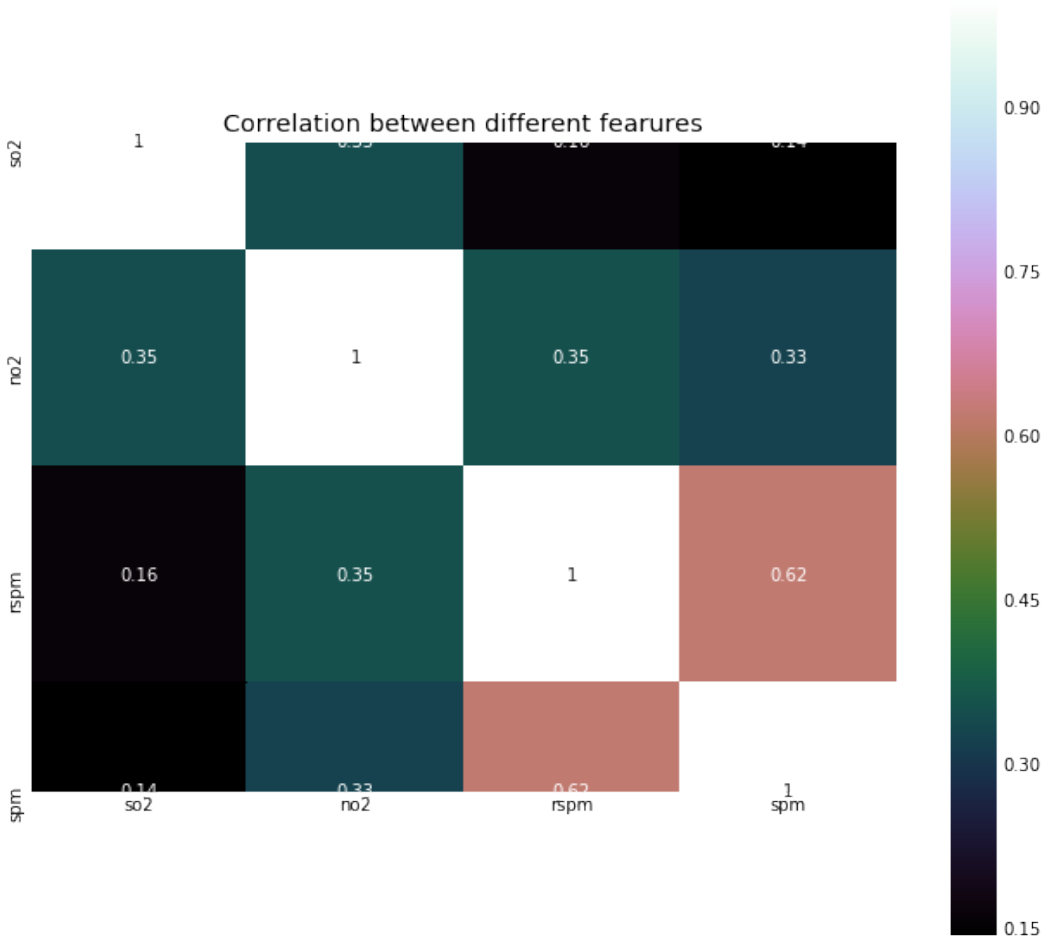
df.corr() compute pairwise correlation of columns.Correlation shows how the two variables are related to each other.Positive values shows as one variable increases other variable increases as well. Negative values shows as one variable increases other variable decreases.Bigger the values,more strongly two varibles are correlated and viceversa.

[12]:
```
# Computing coorelation between pollutants
df = dataset[['so2','no2','rspm','spm']]   # independent variables matrix -␣
 ↪pollutant concentrations
df.corr() # computing correlation
```

[12]:
```
           so2       no2      rspm       spm
so2   1.000000  0.346429  0.160417  0.142689
no2   0.346429  1.000000  0.352189  0.326767
rspm  0.160417  0.352189  1.000000  0.618619
spm   0.142689  0.326767  0.618619  1.000000
```

[13]:
```
correlation = df.corr() # computing correlation
plt.figure(figsize=(10,10)) # plotting results
sns.heatmap(correlation, vmax=1, square=True, annot=True, cmap='cubehelix')

plt.title('Correlation between different fearures')
```

[13]:
```
Text(0.5, 1, 'Correlation between different fearures')
```



[14]:
```
# Derivation for Individual Pollutant Index and AQI STARTS HERE

# EPA METHOD FORMULA
#  AQI_{P} = AQI_{min} +  ({PM_{Obs}-PM_{Min}}/{(PM_{Max}-PM_{Min})})*␣
 ↪{AQI_{Max}-AQI_{Min}}

# calculating AQI index of SO2 pollutant by EPA method formula given above
#  SO2 is scaled between 0-1600
def calculate_si(so2):
    si=0
    if (so2<=40):
     si = so2 * (50/40)
    elif (so2>40 and so2<=80):
     si = 50 + (so2-40) * (50/(80-40))
    elif (so2>80 and so2<=380):
     si = 100 + (so2-80) * (100/(380-80))
    elif (so2>380 and so2<=800):
     si = 200 + (so2-380) * (100/(800-380))
```

```
    elif (so2>800 and so2<=1600):
     si = 300 + (so2-800) * (100/(1600-800))
    elif (so2>1600):
     si = 400 + (so2-1600) * (100/800)
    return si

# calling the function to calulate so2 pollutant index
dataset['si'] = dataset['so2'].apply(calculate_si)
df_si = dataset[['so2','si']]
df_si.head()
```

[14]:

	so2	si
0	4.8	6.000
1	3.1	3.875
2	6.2	7.750
3	6.3	7.875
4	4.7	5.875

[15]:
```
#Function to calculate no2 individual pollutant index(ni)

# EPA METHOD FORMULA
#  AQI_{PM} = AQI_{min} +  ({PM_{Obs}-PM_{Min}}/{(PM_{Max}-PM_{Min})})*␣
 ↪{AQI_{Max}-AQI_{Min}}

# calculating AQI index of NO2 pollutant by EPA method formula given above
#  NO2 is scaled between 0-400
def calculate_ni(no2):
    ni = 0
    if(no2<=40):
     ni = no2*50/40
    elif(no2>40 and no2<=80):
     ni = 50 + (no2-40)*(50/(80-40))
    elif(no2>80 and no2<=180):
     ni = 100 + (no2-80)*(100/(180-80))
    elif(no2>180 and no2<=280):
     ni = 200 + (no2-180)*(100/(280-180))
    elif(no2>280 and no2<=400):
     ni = 300 + (no2-280)*(100/(400-280))
    else:
     ni = 400 + (no2-400)*(100/120)
    return ni

# calling the function to calulate so2 pollutant index
dataset['ni'] = dataset['no2'].apply(calculate_ni)
df_ni = dataset[['no2','ni']]
df_ni.head()
```

[15]:

	no2	ni
0	17.4	21.750
1	7.0	8.750
2	28.5	35.625
3	14.7	18.375
4	7.5	9.375

[16]:
```
#Function to calculate rspm individual pollutant index(rpi)

# EPA METHOD FORMULA
#  AQI_{PM} = AQI_{min} +  ({PM_{Obs}-PM_{Min}}/{(PM_{Max}-PM_{Min})})*␣
 ↪{AQI_{Max}-AQI_{Min}}

# calculating AQI index of RSPM pollutant by EPA method formula given above
#  RSPM is scaled between 0-400
def calculate_(rspm):
    rpi=0
    if(rpi<=30):
     rpi = rpi*50/30
    elif(rpi>30 and rpi<=60):
     rpi = 50+(rpi-30)*50/(60-30)
    elif(rpi>60 and rpi<=90):
     rpi = 100+(rpi-60)*100/(90-60)
    elif(rpi>90 and rpi<=120):
     rpi = 200+(rpi-90)*100/(120-90)
    elif(rpi>120 and rpi<=250):
     rpi = 300+(rpi-120)*(100/(250-120))
    else:
     rpi = 400+(rpi-250)*(100/130)
    return rpi

# calling the function to calulate RSPM pollutant index
dataset['rpi']=dataset['rspm'].apply(calculate_si)
df_rpi = dataset[['rspm','rpi']]
df_rpi.head()
```

[16]:

	rspm	rpi
0	78.182824	97.72853
1	78.182824	97.72853
2	78.182824	97.72853
3	78.182824	97.72853
4	78.182824	97.72853

[17]:
```
#Function to calculate spm individual pollutant index(spi)

# EPA METHOD FORMULA
#  AQI_{PM} = AQI_{min} +  ({PM_{Obs}-PM_{Min}}/{(PM_{Max}-PM_{Min})})*␣
 ↪{AQI_{Max}-AQI_{Min}}
```

```
# calculating AQI index of SPM pollutant by EPA method formula given above
#  SPM is scaled between 0-400
def calculate_spi(spm):
    spi=0
    if(spm<=50):
     spi = spm*50/50
    elif(spm>50 and spm<=100):
     spi = 50 + (spm-50)*(50/(100-50))
    elif(spm>100 and spm<=250):
     spi = 100 + (spm-100)*(100/(250-100))
    elif(spm>250 and spm<=350):
     spi=200 + (spm-250)*(100/(350-250))
    elif(spm>350 and spm<=430):
     spi=300 + (spm-350)*(100/(430-350))
    else:
     spi=400+(spm-430)*(100/430)
    return spi

# calling the function to calulate SPM pollutant index
dataset['spi'] = dataset['spm'].apply(calculate_spi)
df_spm = dataset[['spm','spi']]
df_spm.head()
```

[17]:

	spm	spi
0	200.260378	166.840252
1	200.260378	166.840252
2	200.260378	166.840252
3	200.260378	166.840252
4	200.260378	166.840252

[19]:
```
#function to calculate the air quality index (AQI) of every data value its␣
 ↪is calculated as per indian govt standards

# AQI = MAX ( AQI_{SO2}, AQI_{NO2}, AQI_{RSPM}, AQI_{SPM})
def calculate_aqi(si,ni,spi,rpi):
    aqi=0
    if(si>ni and si>spi and si>rpi):
     aqi=si
    if(spi>si and spi>ni and spi>rpi):
     aqi=spi
    if(ni>si and ni>spi and ni>rpi):
     aqi=ni
    if(rpi>si and rpi>ni and rpi>spi):
     aqi=rpi
    return aqi

# calling the function to calulate AQI
```

```
dataset['AQI'] = dataset.apply(lambda x:
 ↪calculate_aqi(x['si'],x['ni'],x['spi'],x['rpi']),axis=1)
df= dataset[['state','si','ni','rpi','spi','AQI']]
df.head()
```

[19]:

	state	si	ni	rpi	spi	AQI
0	Andhra Pradesh	6.000	21.750	97.72853	166.840252	166.840252
1	Andhra Pradesh	3.875	8.750	97.72853	166.840252	166.840252
2	Andhra Pradesh	7.750	35.625	97.72853	166.840252	166.840252
3	Andhra Pradesh	7.875	18.375	97.72853	166.840252	166.840252
4	Andhra Pradesh	5.875	9.375	97.72853	166.840252	166.840252

[20]:
```
# filling 0 in the place of NAN values
dataset.fillna(0.0, inplace=True)
state=dataset.groupby(['state'],as_index=False).mean()  # calulating mean␣
 ↪of the pollutant concentration for each state
```

Computing Correlation among the features in the dataset

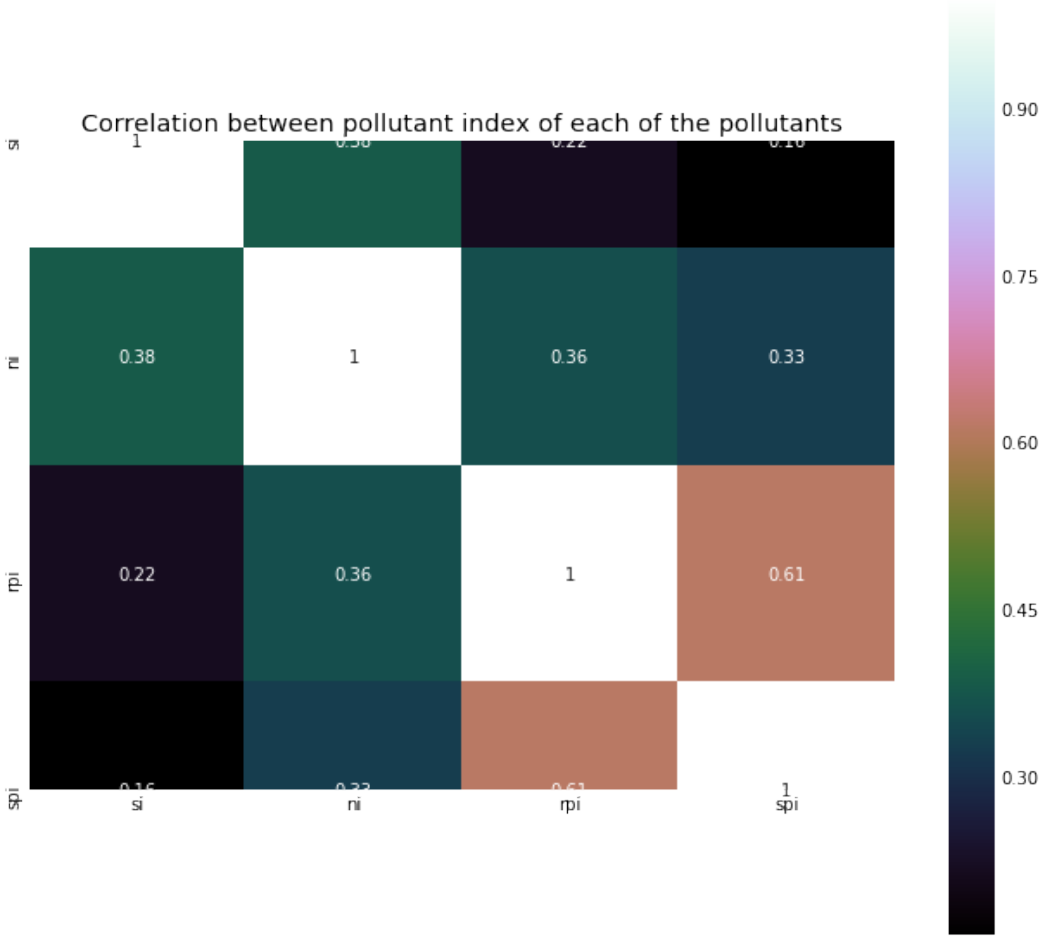
[21]:
```
# Computing coorelation between pollutants
df1 = dataset[['si','ni','rpi','spi']]   # independent variables matrix -␣
 ↪pollutant concentrations
df1.corr()
```

[21]:

	si	ni	rpi	spi
si	1.000000	0.383492	0.215005	0.158088
ni	0.383492	1.000000	0.360968	0.328635
rpi	0.215005	0.360968	1.000000	0.613163
spi	0.158088	0.328635	0.613163	1.000000

[22]:
```
correlation = df1.corr()
plt.figure(figsize=(10,10))
sns.heatmap(correlation, vmax=1, square=True, annot=True, cmap='cubehelix')

plt.title('Correlation between pollutant index of each of the pollutants')
```

[22]:
```
Text(0.5, 1, 'Correlation between pollutant index of each of the␣
 ↪pollutants')
```



1 2 PCA Analysis

1.1 2.1 Functions for computing PCA

1.1.1 2.1.1 Normalize Function

[23]:
```
# Normalize Function
def normalize(X):
  """Normalize the given dataset X
  Args:
  X: ndarray, dataset

  Returns:
  (Xbar, mean, std): tuple of ndarray, Xbar is the normalized dataset
  with mean 0 and standard deviation 1; mean and std are the
  mean and standard deviation respectively.

  Note:
  You will encounter dimensions where the standard deviation is
```

```
    zero, for those when you do normalization the normalized data
    will be NaN. Handle this by setting using `std = 1` for those
    dimensions when doing normalization.
    """
    mu = np.mean(X, axis=0) #mean
    std = np.std(X, axis=0) #standard deviation
    new_std = np.where(std>0,std,1) #new standard deviation
    a = np.subtract(X,mu) # a = x- mean
    Xbar = np.divide(a,new_std) # Xbar = a/new standard deviation
    return Xbar, mu, std
```

1.1.2 2.1.2 Eig Function

Computes Eigen Values and Eigen Vectors of Covariance Matrix S

[24]:
```
def eig(S):
    """Compute the eigenvalues and corresponding eigenvectors
    for the covariance matrix S.
    Args:
    S: ndarray, covariance matrix
    Returns:
    (eigvals, eigvecs): ndarray, the eigenvalues and eigenvectors
    Note:
    the eigenvals and eigenvecs should be sorted in descending
    order of the eigen values
    """
    eigvals, eigvecs = np.linalg.eig(S) # computing eigen values and eigen␣
    ↪vectors of our feature matrix
    idx = eigvals.argsort()[::-1]       # sorting values in descending order
    eigvals = eigvals[idx]              # replacing by sorted array
    eigvecs = eigvecs[:,idx]            # replacing by sorted array

    return (eigvals, eigvecs)
```

1.1.3 2.1.3 Projection Matrix Function

[25]:
```
def projection_matrix(B):
    """Compute the projection matrix onto the space spanned by `B`
    Args:
    B: ndarray of dimension (D, M), the basis for the subspace

    Returns:
    P: the projection matrix
    """
    P = np.matmul(B,B.T) # P = B*B.T (projection Matrix)
    return P
```

1.1.4 2.1.4 PCA Function

Takes X (input matrix) and M (number of components) as input and Computes Reconstructed X (input) matrix by appling Principal Component Analysis (PCA)

[26]:
```
def PCA(X, num_components):
  """
  Args:
  X: ndarray of size (N, D), where D is the dimension of the data,
  and N is the number of datapoints
  num_components: the number of principal components to use.
  Returns:
  X_reconstruct: ndarray of the reconstruction
  of X from the first `num_components` principal components.
  """
  Xbar, mu, std = normalize(X) # normalizing data
  covariance = np.dot(Xbar.T,Xbar) # computing covariancematrix
  S = covariance # S is denoted as our covariance matrix
  eigvals, eigvecs = eig(S) # eigvals = eigen values and eigvecs = eigen␣
↪vectors

  sum_value = sum(eigvals[:num_components]) # sum of M (num_components)␣
↪eigen values (principal components)
  B = np.stack(eigvecs[:,:num_components]) # taking first M␣
↪(num_components) principal components to form matrix B
  P = np.matmul(B,B.T) # calculating P (projection matrix)  for  B
  X_reconstruct = np.matmul(P,X.T) # reconstructing X
  X_reconstruct = X_reconstruct.T # transposing reconstructed X
  return X_reconstruct, sum_value
```

1.2 2.2 Analysis of PCA for a particular value of M (say M=2)

1.2.1 2.2.1 Defining Input Matrix

[27]:
```
# Computing input matrix X having four features and 435742 rows
X = dataset[['si','ni','rpi','spi']]   # independent variables matrix -␣
↪pollutant concentrations
y = dataset['AQI']                     # target variable matrix - AQI

print('The dimension of input matrix X is: ', X.shape)   # print shape of␣
↪input matrix X
print(X)
```

```
The dimension of input matrix X is:  (435742, 4)
            si      ni         rpi         spi
0        6.000  21.750   97.728530  166.840252
1        3.875   8.750   97.728530  166.840252
2        7.750  35.625   97.728530  166.840252
3        7.875  18.375   97.728530  166.840252
```

4	5.875	9.375	97.728530	166.840252
5	8.000	32.125	97.728530	166.840252
6	6.750	21.375	97.728530	166.840252
7	5.875	10.875	97.728530	166.840252
8	5.250	28.750	97.728530	166.840252
9	5.000	11.125	97.728530	166.840252
10	4.500	23.250	97.728530	166.840252
11	4.875	17.625	97.728530	122.000000
12	7.000	14.750	97.728530	82.000000
13	4.125	24.125	97.728530	107.333333
14	4.875	10.250	97.728530	112.000000
15	4.375	15.125	97.728530	123.333333
16	9.875	12.750	97.728530	80.000000
17	5.000	12.375	97.728530	152.666667
18	15.500	14.375	97.728530	58.000000
19	5.000	15.375	97.728530	99.000000
20	7.875	14.375	97.728530	220.000000
21	56.000	17.125	97.728530	97.000000
22	10.125	22.250	97.728530	144.666667
23	9.625	14.125	97.728530	130.000000
24	25.750	17.000	97.728530	75.000000
25	25.500	34.375	97.728530	174.666667
26	17.375	9.000	97.728530	93.000000
27	14.000	23.250	97.728530	61.000000
28	27.875	44.875	97.728530	205.000000
29	30.625	35.000	97.728530	164.666667
...	...	...	...	...
435712	15.000	47.500	107.666667	189.004349
435713	17.500	45.000	111.000000	189.004349
435714	25.000	51.250	117.000000	189.004349
435715	12.500	47.500	102.666667	189.004349
435716	15.000	50.000	112.000000	189.004349
435717	26.250	65.000	118.000000	189.004349
435718	26.250	61.250	132.666667	189.004349
435719	22.500	58.750	119.333333	189.004349
435720	23.750	56.250	115.333333	189.004349
435721	13.750	42.500	103.333333	189.004349
435722	20.000	50.000	118.333333	189.004349
435723	18.750	46.250	118.666667	189.004349
435724	27.500	83.750	140.666667	189.004349
435725	25.000	80.000	133.666667	189.004349
435726	17.500	52.500	105.000000	189.004349
435727	13.750	42.500	112.666667	189.004349
435728	21.250	53.750	121.333333	189.004349
435729	23.750	61.250	120.000000	189.004349
435730	22.500	51.250	120.666667	189.004349
435731	27.500	72.500	125.000000	189.004349
435732	27.500	62.500	121.666667	189.004349
435733	42.500	76.250	127.000000	189.004349

```
435734  25.000  55.000  122.666667  189.004349
435735  21.250  55.000  117.000000  189.004349
435736  22.500  56.250  120.000000  189.004349
435737  27.500  62.500  121.000000  189.004349
435738  25.000  57.500  130.333333  189.004349
435739   0.000   0.000    0.000000    0.000000
435740   0.000   0.000    0.000000    0.000000
435741   0.000   0.000    0.000000    0.000000

[435742 rows x 4 columns]
```

1.2.2 2.2.2 Normalizing the Dataset

Normalization refers to shifting the distribution of each attribute to have a mean of zero and a standard deviation of one (unit variance). It is useful to standardize attributes for a model. Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data

[28]:
```
## Some preprocessing of the data
Xbar, mu, std = normalize(X)          # Normalizing input matrix X
print('The dimension of normalized input matrix X is: ', Xbar.shape)    #␣
 ↪print shape of normalized X (Xbar)
print('The dimension of mean of input matrix X is: ', mu.shape)          #␣
 ↪print shape of mean of X
print('The dimension of standard deviation of input matrix X is: ', X.
 ↪shape)  # print shape of standard deviation of X
```

```
The dimension of normalized input matrix X is:  (435742, 4)
The dimension of mean of input matrix X is:  (4,)
The dimension of standard deviation of input matrix X is:  (435742, 4)
```

1.2.3 2.2.3 Finding the Covariance Matrix

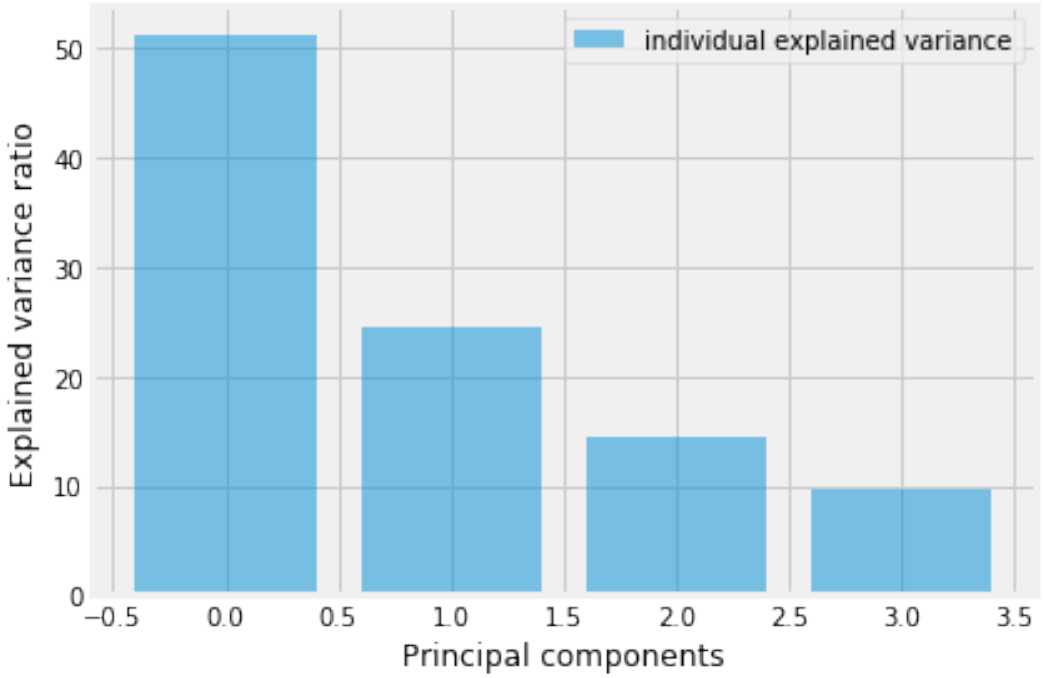
[29]:
```
# Finding Covariance matrix
covariance = np.dot(Xbar.T,Xbar)
print('Shape of covariance matrix : ', covariance.shape)        # print␣
 ↪shape of covariance matrix
print('Trace of covariance matrix : ', np.trace(covariance))    # print␣
 ↪trace of convariance matrix

print()
S = covariance
eigvals, eigvecs = eig(S)      # find eigen values and eigen vectors in␣
 ↪sorted order (decreasing) of covariance matrix
```

```
Shape of covariance matrix :  (4, 4)
Trace of covariance matrix :  1742968.000000151
```

1.2.4 2.2.4 Variance of each principal components/features

Explained Variance After sorting the eigenpairs, the next question is "how many principal components are we going to choose for our new feature subspace?" A useful measure is the so-called "explained variance," which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components.

[30]:
```
tot = sum(eigvals) # sum of all eigenvalues
var_exp = [(i / tot)*100 for i in eigvals] # computing explained variance-␣
 ↪variance contributed to each of the principal components (in %)

plt.figure(figsize=(6, 4)) # mentioning size of the bar graph

#plotting bar graph
plt.bar(range(4), var_exp, alpha=0.5, align='center', label='individual␣
 ↪explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.tight_layout()
```



The above plot above clearly shows that maximum variance (somewhere around 50%) can be explained by the first principal component alone. The second principal component shows variance of about 25%, and Comparatively third and fourth components share less amount

of information as compared to the rest of the Principal components. Hence, for analysis we take M = 2 (number of principal components to be considered for low dimenional data) as they have much more variance than the last two principal components.

[31]:
```
num_components = 2                              # taking random value of M for␣
 ↪analysis
print('Eigen values sorted in decreasing order:')  # printing eigenvalues␣
 ↪in decreasing order
for i in eigvals:
    print(i)
print()
print('Top M (where M=2) Eigenvalues : ', eigvals[:num_components])        ␣
 ↪    # printing top M eigenvalues
print()
print('M eigenvectors correspondong to top M eigenvalues: \n', eigvecs[:,:
 ↪num_components])  # printing top M eigenvectors corresponding to top M␣
 ↪eigenvalues

print()
print('Shape of one eigenvector : ', eigvecs.shape[0])    # printing␣
 ↪eigenvector shape
```

```
Eigen values sorted in decreasing order:
894576.628509095
428403.78785137954
252699.9521340215
167287.6315056545

Top M (where M=2) Eigenvalues :  [894576.6285091  428403.78785138]

M eigenvectors correspondong to top M eigenvalues:
 [[ 0.37771276  0.72301441]
 [ 0.49901073  0.36349529]
 [ 0.56268523 -0.36866975]
 [ 0.54009878 -0.45738818]]

Shape of one eigenvector :  4
```

1.2.5 2.2.5 Plotting sorted eigenvalues in decreasing order

[32]:
```
# Plotting sorted eigen values (in decreasing order) v/s its index
eigvals = np.asarray(eigvals)       # converting to array
eig_df = pd.DataFrame(eigvals)      # making a dataframe with index and␣
 ↪eigenvalues
eig_df.rename(columns={0 : 'eigenvalues'},  inplace = True)  #renaming the␣
 ↪column of dataframe to eigenvalues
print(eig_df.head())        # printing all the eigenvalues ( 1 to D (total␣
 ↪number of features i.e. 4))
```

```
fig, ax = plt.subplots()
ax.plot(eigvals);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='index', ylabel='Eigen values', title='Eigen Values vs index␣
 ↪(decreasing order)');
```

```
   eigenvalues
0  894576.628509
1  428403.787851
2  252699.952134
3  167287.631506
```

1.2.6 2.2.6 Calculating Projection Matrix and calling PCA function

[33]:
```
## Calculation of Projection matrix and Reconstructed X matrix
B = np.stack(eigvecs[:,:num_components])
print('Shape of matrix B (top M eigenvectors (M=2 in this case)) : ', B.
 ↪shape)
print('Trace of matrix B : ', np.trace(B))

print()
P = projection_matrix(B)  # finding projection matrix P
print('Shape of Projection matrix : ', P.shape)

print()
X_reconstruct, sum_value = PCA(Xbar, num_components)            # finding␣
 ↪reconstructed X by selecting num_components principal components
```

```
print('Shape of X_reconstruct matrix : ', X_reconstruct.shape)  # printing␣
 ↪shape of X_reconstruct matrix
print('Shape of normalized X matrix : ', Xbar.shape)            # printing␣
 ↪shape of normalized X matrix

print()
print('X reconstruced : \n', X_reconstruct)                     # printing␣
 ↪projected data matrix (reconstructed X)
```

```
Shape of matrix B (top M eigenvectors (M=2 in this case)) :  (4, 2)
Trace of matrix B :  0.741208053351313

Shape of Projection matrix :  (4, 4)

Shape of X_reconstruct matrix :  (435742, 4)
Shape of normalized X matrix :  (435742, 4)

X reconstruced :
                si        ni       rpi       spi
0      -0.580874 -0.465445 -0.127513 -0.069566
1      -0.964828 -0.770251 -0.204838 -0.108370
2      -0.199599 -0.159465 -0.042668 -0.022720
3      -0.546948 -0.453963 -0.158435 -0.105079
4      -0.841879 -0.684731 -0.209604 -0.126399
5      -0.257708 -0.210830 -0.067161 -0.041786
6      -0.547282 -0.443943 -0.133370 -0.079190
7      -0.811076 -0.658716 -0.199586 -0.119351
8      -0.478421 -0.372049 -0.077410 -0.028815
9      -0.854117 -0.687054 -0.194005 -0.109004
10     -0.632657 -0.495441 -0.110791 -0.046794
11     -0.665787 -0.629306 -0.380282 -0.321230
12     -0.552758 -0.644700 -0.614376 -0.574744
13     -0.553409 -0.561039 -0.408829 -0.362662
14     -0.803466 -0.768430 -0.480886 -0.410314
15     -0.746489 -0.689837 -0.387898 -0.320477
16     -0.432785 -0.574276 -0.650853 -0.625166
17     -0.808935 -0.681279 -0.258436 -0.180281
18     -0.059429 -0.360738 -0.778109 -0.796249
19     -0.673443 -0.689467 -0.513970 -0.458307
20     -0.702277 -0.463687  0.087818  0.165489
21      2.173168  1.243011 -0.740505 -0.995602
22     -0.312968 -0.327623 -0.256469 -0.231155
23     -0.487152 -0.503661 -0.383810 -0.343923
24      0.535408  0.086604 -0.719099 -0.798831
25      0.741224  0.490434 -0.090166 -0.172065
26     -0.114760 -0.344670 -0.642668 -0.650645
27      0.036104 -0.259464 -0.696726 -0.722496
28      1.045842  0.795254  0.125102  0.017484
29      1.049994  0.681423 -0.160247 -0.277286
```

```
...          ...       ...       ...       ...
435712  0.398123  0.382239  0.241889  0.207034
435713  0.479470  0.445706  0.255557  0.212454
435714  1.011819  0.858404  0.338545  0.241270
435715  0.267918  0.268678  0.190756  0.168180
435716  0.443015  0.443111  0.312586  0.275166
435717  1.361511  1.147587  0.437251  0.305782
435718  1.262687  1.141831  0.594974  0.479008
435719  1.024718  0.904556  0.428886  0.333077
435720  1.048152  0.891707  0.356756  0.256179
435721  0.233072  0.231334  0.160083  0.140259
435722  0.708880  0.655412  0.369160  0.305161
435723  0.562556  0.545047  0.353856  0.304983
435724  1.781648  1.611057  0.839349  0.675717
435725  1.577411  1.424377  0.738248  0.593220
435726  0.642409  0.551527  0.230878  0.169619
435727  0.219188  0.269057  0.276390  0.261706
435728  0.850246  0.779249  0.426002  0.348713
435729  1.143886  0.997283  0.448303  0.340395
435730  0.868721  0.779873  0.395411  0.315189
435731  1.573933  1.352628  0.568985  0.419002
435732  1.373540  1.165726  0.460659  0.328644
435733  2.473824  1.985857  0.551907  0.305401
435734  1.080396  0.946344  0.434205  0.332625
435735  0.882361  0.783413  0.380351  0.298200
435736  0.972388  0.863893  0.420497  0.330006
435737  1.374532  1.163031  0.452352  0.319969
435738  1.120329  1.020688  0.546439  0.444131
435739 -0.982780 -1.648900 -2.320495 -2.288670
435740 -0.982780 -1.648900 -2.320495 -2.288670
435741 -0.982780 -1.648900 -2.320495 -2.288670

[435742 rows x 4 columns]
```

[34]:
```
## Calculation of unnormalized X reconstructed matrix
mu = np.mean(X, axis=0)    # finding mean of input matrix X
std = np.std(X, axis=0)    # finding standard deviation of input matrix X
X_final = X_reconstruct * std + mu    # unnomalizing X reconstruct to␣
 ↪X_final
print('Shape of unnormalized reconstructed matrix : ', X_reconstruct.shape)␣
 ↪ # printing shape of unnormalized reconstructed matrix
print('Shape of input matrix X : ', Xbar.shape)      # printing shape of␣
 ↪input matrix X matrix
```

```
Shape of unnormalized reconstructed matrix :  (435742, 4)
Shape of input matrix X :  (435742, 4)
```

1.2.7 2.2.7 Comparison of our PCA function with sklearn library

[35]:
```
# Comparison user functions with inbuilt Library
from sklearn.preprocessing import StandardScaler

standardized_data = StandardScaler(with_mean=True, with_std=True)
sample_data = standardized_data.fit_transform(Xbar)     # normalizing the␣
 ↪dataset using library

print('Shape of Normalized input matrix computed using library : ' ,␣
 ↪sample_data.shape)
print('Shape of Normalized input matrix computed using user functions : ' ,␣
 ↪Xbar.shape)

# Find the co-variance matrix which is : A^T * A

# matrix multiplication using numpy
covar_matrix = np.matmul(sample_data.T, sample_data)  # finding covariance␣
 ↪matrix using normalized X (using inbuilt library)
print()
print("Shape of covariance matrix computed using Library = ", covar_matrix.
 ↪shape)
print("Shape of covariance matrix computed by user functions = ", S.shape)

print()
print('The covariance matrix is : \n' ,covar_matrix)
print()
print('The trace of covariance matrix : ', np.trace(covar_matrix))
```

```
Shape of Normalized input matrix computed using library :  (435742, 4)
Shape of Normalized input matrix computed using user functions :  (435742, 4)

Shape of covariance matrix computed using Library =  (4, 4)
Shape of covariance matrix computed by user functions =  (4, 4)

The covariance matrix is :
 [[435742.         167103.51206591  93686.88467662  68885.58764466]
 [167103.51206591 435742.         157288.71880688 143200.16261006]
 [ 93686.88467662 157288.71880688 435742.         267180.71513081]
 [ 68885.58764466 143200.16261006 267180.71513081 435742.00000001]]

The trace of covariance matrix :  1742968.000000007
```

1.2.8 2.2.8 Comparison of Actual AQI and Reconstructed AQI formed from projected data matrix X_tilda

[36]:
```
# Computing Reconstructed AQI from unnormalized reconstructed X matrix
dataset['AQI_reconstructed'] = X_final.apply(lambda x:
 ↪calculate_aqi(x['si'],x['ni'],x['spi'],x['rpi']),axis=1)

# Making Reconstruced Columns for all the pollutants
dataset['ni_reconstructed'] =  X_final['ni'] # Making Reconstruced Columns␣
 ↪for no2
dataset['si_reconstructed'] =  X_final['si'] # Making Reconstruced Columns␣
 ↪for so2
dataset['rpi_reconstructed'] =  X_final['rpi'] # Making Reconstruced␣
 ↪Columns for rspm
dataset['spi_reconstructed'] =  X_final['spi'] # Making Reconstruced␣
 ↪Columns for spm

# Defining a dataframe consisting of Actual AQI and pollutant values and␣
 ↪Reconstructed AQI and pollutant values
df =␣
 ↪dataset[['state','si','ni','spi','rpi','si_reconstructed','ni_reconstructed','rpi_recons
df
```

[36]:

	state	si	ni	spi	rpi ␣ ↪ \
0	Andhra Pradesh	6.000	21.750	166.840252	97.728530
1	Andhra Pradesh	3.875	8.750	166.840252	97.728530
2	Andhra Pradesh	7.750	35.625	166.840252	97.728530
3	Andhra Pradesh	7.875	18.375	166.840252	97.728530
4	Andhra Pradesh	5.875	9.375	166.840252	97.728530
5	Andhra Pradesh	8.000	32.125	166.840252	97.728530
6	Andhra Pradesh	6.750	21.375	166.840252	97.728530
7	Andhra Pradesh	5.875	10.875	166.840252	97.728530
8	Andhra Pradesh	5.250	28.750	166.840252	97.728530
9	Andhra Pradesh	5.000	11.125	166.840252	97.728530
10	Andhra Pradesh	4.500	23.250	166.840252	97.728530
11	Andhra Pradesh	4.875	17.625	122.000000	97.728530
12	Andhra Pradesh	7.000	14.750	82.000000	97.728530
13	Andhra Pradesh	4.125	24.125	107.333333	97.728530
14	Andhra Pradesh	4.875	10.250	112.000000	97.728530
15	Andhra Pradesh	4.375	15.125	123.333333	97.728530
16	Andhra Pradesh	9.875	12.750	80.000000	97.728530
17	Andhra Pradesh	5.000	12.375	152.666667	97.728530
18	Andhra Pradesh	15.500	14.375	58.000000	97.728530
19	Andhra Pradesh	5.000	15.375	99.000000	97.728530
20	Andhra Pradesh	7.875	14.375	220.000000	97.728530
21	Andhra Pradesh	56.000	17.125	97.000000	97.728530
22	Andhra Pradesh	10.125	22.250	144.666667	97.728530

23	Andhra Pradesh	9.625	14.125	130.000000	97.728530
24	Andhra Pradesh	25.750	17.000	75.000000	97.728530
25	Andhra Pradesh	25.500	34.375	174.666667	97.728530
26	Andhra Pradesh	17.375	9.000	93.000000	97.728530
27	Andhra Pradesh	14.000	23.250	61.000000	97.728530
28	Andhra Pradesh	27.875	44.875	205.000000	97.728530
29	Andhra Pradesh	30.625	35.000	164.666667	97.728530
...	...	...	...	...	...
435712	West Bengal	15.000	47.500	189.004349	107.666667
435713	West Bengal	17.500	45.000	189.004349	111.000000
435714	West Bengal	25.000	51.250	189.004349	117.000000
435715	West Bengal	12.500	47.500	189.004349	102.666667
435716	West Bengal	15.000	50.000	189.004349	112.000000
435717	West Bengal	26.250	65.000	189.004349	118.000000
435718	West Bengal	26.250	61.250	189.004349	132.666667
435719	West Bengal	22.500	58.750	189.004349	119.333333
435720	West Bengal	23.750	56.250	189.004349	115.333333
435721	West Bengal	13.750	42.500	189.004349	103.333333
435722	West Bengal	20.000	50.000	189.004349	118.333333
435723	West Bengal	18.750	46.250	189.004349	118.666667
435724	West Bengal	27.500	83.750	189.004349	140.666667
435725	West Bengal	25.000	80.000	189.004349	133.666667
435726	West Bengal	17.500	52.500	189.004349	105.000000
435727	West Bengal	13.750	42.500	189.004349	112.666667
435728	West Bengal	21.250	53.750	189.004349	121.333333
435729	West Bengal	23.750	61.250	189.004349	120.000000
435730	West Bengal	22.500	51.250	189.004349	120.666667
435731	West Bengal	27.500	72.500	189.004349	125.000000
435732	West Bengal	27.500	62.500	189.004349	121.666667
435733	West Bengal	42.500	76.250	189.004349	127.000000
435734	West Bengal	25.000	55.000	189.004349	122.666667
435735	West Bengal	21.250	55.000	189.004349	117.000000
435736	West Bengal	22.500	56.250	189.004349	120.000000
435737	West Bengal	27.500	62.500	189.004349	121.000000
435738	West Bengal	25.000	57.500	189.004349	130.333333
435739	andaman-and-nicobar-islands	0.000	0.000	0.000000	0.000000
435740	Lakshadweep	0.000	0.000	0.000000	0.000000
435741	Tripura	0.000	0.000	0.000000	0.000000

	si_reconstructed	ni_reconstructed	rpi_reconstructed	\
0	6.117200	21.731572	94.545145	
1	1.476775	15.032918	91.737112	
2	10.725256	28.455989	97.626264	
3	6.527229	21.983893	93.422250	
4	2.962724	16.912380	91.564067	
5	10.022960	27.327153	96.736813	
6	6.523193	22.204112	94.332458	
7	3.335003	17.484090	91.927869	

8	7.355437	23.784087	96.364624
9	2.814810	16.861329	92.130528
10	5.491353	21.072346	95.152419
11	5.090958	18.130442	85.365943
12	6.457013	17.792124	76.864947
13	6.449148	19.630720	84.329265
14	3.426983	15.072939	81.712551
15	4.115599	16.800158	85.089389
16	7.906994	19.339819	75.540283
17	3.360883	16.988246	89.790751
18	12.419333	24.032683	70.919048
19	4.998425	16.808294	80.511146
20	4.649937	21.770205	102.364796
21	39.402284	59.277810	72.284595
22	9.355093	24.760435	89.862161
23	7.249922	20.891689	85.237834
24	19.608472	33.863781	73.061962
25	22.095944	42.738637	95.901400
26	11.750610	24.385793	75.837525
27	13.573938	26.258346	73.874425
28	25.777529	49.437590	103.718766
29	25.827716	46.935958	93.356424
...	...	...	...
435712	17.949261	40.360867	107.959841
435713	18.932411	41.755661	108.456158
435714	25.366326	50.825425	111.469833
435715	16.375612	37.865179	106.102968
435716	18.491819	41.698628	110.527151
435717	29.592672	57.180713	115.054324
435718	28.398291	57.054210	120.781955
435719	25.522227	51.839677	114.750538
435720	25.805448	51.557304	112.131180
435721	15.954469	37.044473	104.989089
435722	21.705039	46.364311	112.581614
435723	19.936578	43.938863	112.025846
435724	34.670412	67.366256	129.656350
435725	32.202023	63.263640	125.984907
435726	20.901674	44.081262	107.559976
435727	15.786669	37.873511	109.212721
435728	23.413570	49.085845	114.645822
435729	26.962477	53.877526	115.455669
435730	23.636863	49.099557	113.534903
435731	32.159986	61.686826	119.838165
435732	29.738057	57.579335	115.904378
435733	43.035977	75.603118	119.218015
435734	26.195144	52.758046	114.943686
435735	23.801709	49.177360	112.988018
435736	24.889777	50.946043	114.445889

435737	29.750043	57.520118	115.602690
435738	26.677773	54.391893	119.019435
435739	1.259806	-4.276904	14.907951
435740	1.259806	-4.276904	14.907951
435741	1.259806	-4.276904	14.907951

	spi_reconstructed	AQI	AQI_reconstructed
0	174.691802	166.840252	174.691802
1	171.120923	166.840252	171.120923
2	179.002764	166.840252	179.002764
3	171.423756	166.840252	171.423756
4	169.461780	166.840252	169.461780
5	177.248289	166.840252	177.248289
6	173.806146	166.840252	173.806146
7	170.110339	166.840252	170.110339
8	178.441924	166.840252	178.441924
9	171.062534	166.840252	171.062534
10	176.787392	166.840252	176.787392
11	151.532580	122.000000	151.532580
12	128.203123	97.728530	128.203123
13	147.719827	107.333333	147.719827
14	143.334728	112.000000	143.334728
15	151.601874	123.333333	151.601874
16	123.563080	97.728530	123.563080
17	164.503299	152.666667	164.503299
18	107.819282	97.728530	107.819282
19	138.918212	99.000000	138.918212
20	196.322550	220.000000	196.322550
21	89.473969	97.728530	89.473969
22	159.821666	144.666667	159.821666
23	149.444296	130.000000	149.444296
24	107.581691	97.728530	107.581691
25	165.259426	174.666667	165.259426
26	121.218364	97.728530	121.218364
27	114.606350	97.728530	114.606350
28	182.702492	205.000000	182.702492
29	155.576528	164.666667	155.576528
...	...	...	...
435712	200.145702	189.004349	200.145702
435713	200.644501	189.004349	200.644501
435714	203.296278	189.004349	203.296278
435715	196.570247	189.004349	196.570247
435716	206.415516	189.004349	206.415516
435717	209.232973	189.004349	209.232973
435718	225.173952	189.004349	225.173952
435719	211.744804	189.004349	211.744804
435720	204.668275	189.004349	204.668275
435721	194.000816	189.004349	194.000816

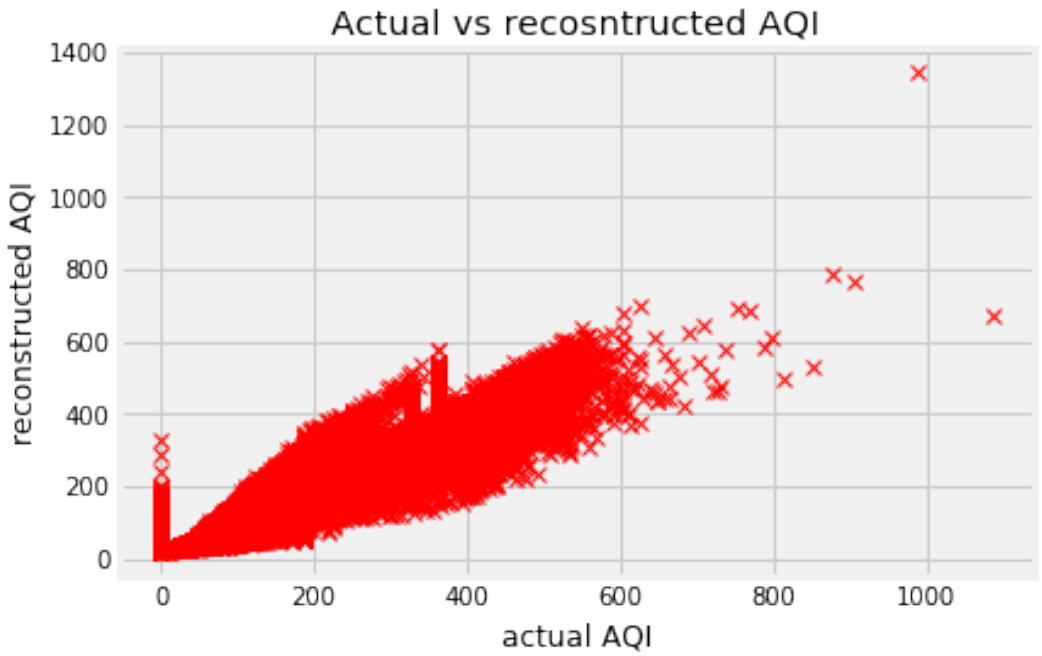
```
435722         209.175831  189.004349         209.175831
435723         209.159439  189.004349         209.159439
435724         243.275947  189.004349         243.275947
435725         235.684227  189.004349         235.684227
435726         196.702685  189.004349         196.702685
435727         205.176873  189.004349         205.176873
435728         213.183672  189.004349         213.183672
435729         212.418165  189.004349         212.418165
435730         210.098591  189.004349         210.098591
435731         219.651948  189.004349         219.651948
435732         211.336776  189.004349         211.336776
435733         209.197898  189.004349         209.197898
435734         211.703138  189.004349         211.703138
435735         208.535253  189.004349         208.535253
435736         211.462163  189.004349         211.462163
435737         210.538486  189.004349         210.538486
435738         221.964401  189.004349         221.964401
435739         -29.519763    0.000000          14.907951
435740         -29.519763    0.000000          14.907951
435741         -29.519763    0.000000          14.907951

[435742 rows x 11 columns]
```

[38]:
```
reconstructed = df['AQI_reconstructed'] #fetching reconstructed data
actual = df['AQI'] # fetching actual data
aqi_reconstruct_error = np.sqrt(np.square(reconstructed - actual).sum()/
 ↪len(actual)) # calculating error between projected data matrix and␣
 ↪original feature matrux

print('Index    Error (AQI recosntructed & Actual)\n', (reconstructed -␣
 ↪actual))
print()
print('Total Error in AQI recosntructed and Actural AQI if taken 2 major␣
 ↪principal components: ' , aqi_reconstruct_error)
print()
fig, ax = plt.subplots() #plotting results
ax.plot(actual, reconstructed, 'rx');
#ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='actual AQI', ylabel='reconstructed AQI', title='Actual vs␣
 ↪recosntructed AQI');
```

```
Index    Error (AQI recosntructed & Actual)
 0          7.851550
1           4.280671
2          12.162512
3           4.583504
4           2.621528
5          10.408037
```

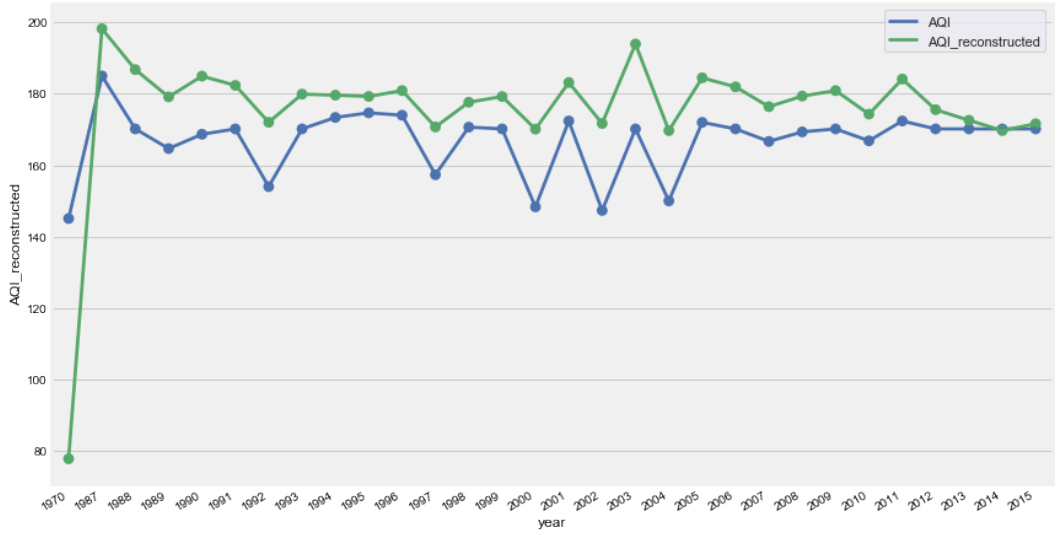
```
6          6.965894
7          3.270087
8         11.601672
9          4.222281
10         9.947140
11        29.532580
12        30.474593
13        40.386493
14        31.334728
15        28.268541
16        25.834550
17        11.836633
18        10.090752
19        39.918212
20       -23.677450
21        -8.254561
22        15.155000
23        19.444296
24         9.853162
25        -9.407240
26        23.489834
27        16.877820
28       -22.297508
29        -9.090139
            ...
435712    11.141352
435713    11.640151
435714    14.291929
435715     7.565898
435716    17.411167
435717    20.228623
435718    36.169603
435719    22.740455
435720    15.663926
435721     4.996467
435722    20.171482
435723    20.155090
435724    54.271598
435725    46.679878
435726     7.698335
435727    16.172524
435728    24.179322
435729    23.413816
435730    21.094242
435731    30.647599
435732    22.332427
435733    20.193549
435734    22.698789
435735    19.530904
```

```
435736    22.457814
435737    21.534137
435738    32.960052
435739    14.907951
435740    14.907951
435741    14.907951
Length: 435742, dtype: float64

Total Error in AQI recosntructed and Actural AQI if taken 2 major principal
components:  38.13322966717384
```



[39]:
```
dataset['date'] = pd.to_datetime(dataset['date'],format='%Y-%m-%d') #␣
 ↪transforming the date column into '%Y-%m-%d' formate
dataset['year'] = dataset['date'].dt.year                           #␣
 ↪extracting year from the date column
dataset['year'] = dataset['year'].fillna(0.0).astype(int)           #␣
 ↪filling nan values as 0
dataset = dataset[(dataset['year']>0)]                              #␣
 ↪extracting only non null values

df = dataset[['AQI','year','state']].groupby(["year"]).median().
 ↪reset_index().sort_values(by='year',ascending=False) #processing fetched␣
 ↪data by grouping on year
```

```
df1 = dataset[['AQI_reconstructed','year','state']].groupby(["year"]).
 ↪median().reset_index().sort_values(by='year',ascending=False) #␣
 ↪processing fetched data by grouping on year
f,ax=plt.subplots(figsize=(13,8))
sns.set()
ax = sns.pointplot(x='year', y='AQI', data=df, label="Actual", color='b',␣
 ↪linestyle="-", errwidth=0.1) #plotting calculated value
ax = sns.pointplot(x='year', y='AQI_reconstructed', data=df1, color='g',␣
 ↪label="predicted", linestyle="-", errwidth=0.1) # plotting predicted␣
 ↪value

# plotting graph
ax.legend(handles=ax.lines[::len(df)+1], labels=["AQI","AQI_reconstructed"])
ax.set_xticklabels([t.get_text().split("T")[0] for t in ax.
 ↪get_xticklabels()])
plt.gcf().autofmt_xdate()
plt.show()
```



1.2.9 2.2.9 Analysis after taking each value of the number of principal component

[40]:
```
for num_component in range(1, 4):
  from sklearn.decomposition import PCA as SKPCA

  # Standard solution given by scikit-learn's implementation of PCA
  pca = SKPCA(n_components=num_component, svd_solver='full')
  sklearn_reconst = pca.inverse_transform(pca.fit_transform(Xbar))

  # Our method
```

```
 reconst, sum_value = PCA(Xbar, num_component) #analysis after taking␣
↪significant components
 np.testing.assert_almost_equal(reconst, sklearn_reconst) # assserting the␣
↪equality between the projected data matrix computed by own method and the␣
↪inbuilt library
 print('Error between reconstructions with sklearn library and our method␣
↪for each feature : \n', np.square(reconst - sklearn_reconst).sum())
 print('Shape of reconstructed matrix : ', reconst.shape)
 print()
```

```
Error between reconstructions with sklearn library and our method for each
feature :
 si     5.951519e-19
ni     5.388616e-19
rpi    2.757037e-19
spi    1.500075e-18
dtype: float64
Shape of reconstructed matrix :  (435742, 4)

Error between reconstructions with sklearn library and our method for each
feature :
 si     1.741128e-19
ni     1.452991e-19
rpi    4.783299e-19
spi    6.454627e-19
dtype: float64
Shape of reconstructed matrix :  (435742, 4)

Error between reconstructions with sklearn library and our method for each
feature :
 si     6.932362e-21
ni     1.762051e-20
rpi    5.853010e-19
spi    5.548941e-19
dtype: float64
Shape of reconstructed matrix :  (435742, 4)
```

1.2.10 2.2.10 Computing loss and variance for every value of M (number of principal components)

[41]:
```
def mse(predict, actual):
  """Helper function for computing the mean squared error (MSE)"""
  return np.square(predict - actual).sum(axis=1).mean()      # returning mse␣
 ↪error
```

[42]:
```
# defining loss, reconstructions, variance_values vector
loss = []
```

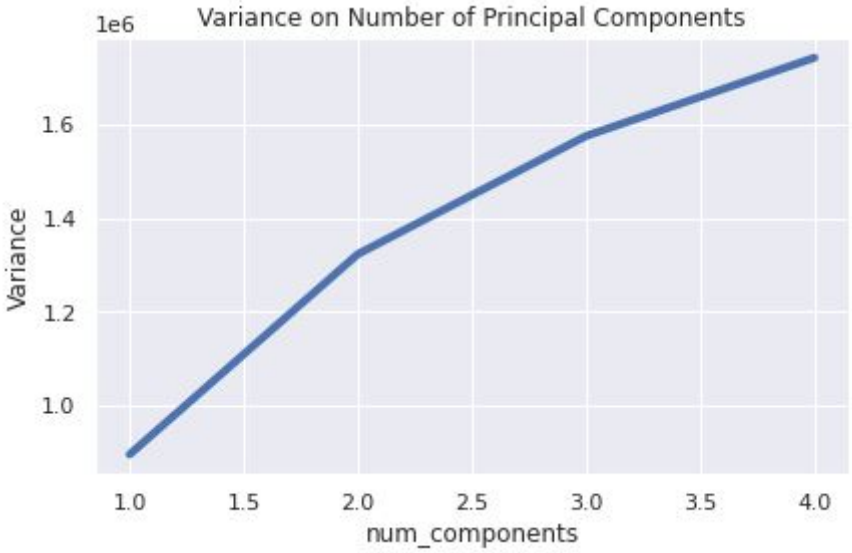
```
reconstructions = []
variance_values = []

X = dataset[['rpi','spi','si','ni']]   # independent variables matrix - pollutant concentrations
Xbar, mu, std = normalize(X)

print('Taking different values of number of components from M=1 to 4 (total number of features available): ')
# iterate over different numbers of principal components, and compute the MSE
for num_component in range(1, 5):
  reconst, sum_value = PCA(Xbar, num_component) #reconst contains reconstructed Xbar, sum value contains summation of significant of num_component
  error = mse(reconst, Xbar)  # computing mse error between projected data matrix and original normalized matrix
  reconst = reconst*std + mu # un-normalizing the data
  reconstructions.append(reconst) # appending un-normalized data
  print('M = {:d}, reconstruction_error = {:f}'.format(num_component,error))
  loss.append((num_component, error)) #appending loss terms
  variance_values.append((num_component, sum_value)) # appending variance
```

```
Taking different values of number of components from M=1 to 4 (total number of features available):
M = 1, reconstruction_error = 1.947004
M = 2, reconstruction_error = 0.963845
M = 3, reconstruction_error = 0.383914
M = 4, reconstruction_error = 0.000000
```

[43]:
```
variance_values = np.asarray(variance_values) # converting variance into variance array
var_df = pd.DataFrame(variance_values) #converting variance into a data frame
var_df.rename(columns={0 : 'num_components', 1: 'Variance'},  inplace = True)  # renaming the columns of dataframe
print(var_df.head())  # printing the varaince values corresponding to all the principal components

# plotting variance plot wrt the number of principal components
sns.set()
fig, ax = plt.subplots()
ax.plot(variance_values[:,0], variance_values[:,1]);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='num_components', ylabel='Variance', title='Variance on Number of Principal Components');
```

```
for l in ax.lines:
    plt.setp(l,linewidth=4)
```

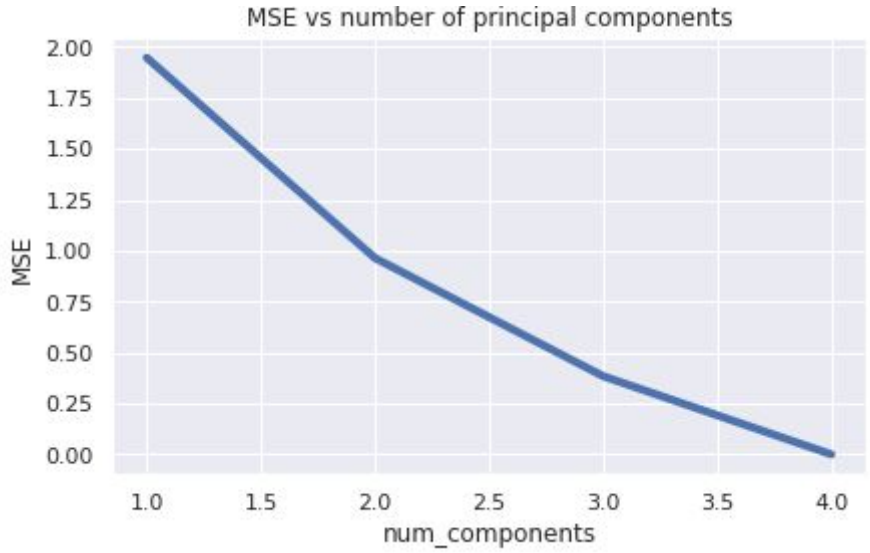
```
   num_components      Variance
0             1.0  8.945766e+05
1             2.0  1.322980e+06
2             3.0  1.575680e+06
3             4.0  1.742968e+06
```



[44]:
```
loss = np.asarray(loss) #converting loss value in array
loss_df = pd.DataFrame(loss) #converting loss into a data frame
loss_df.rename(columns={0 : 'num_components', 1: 'MSE'},  inplace = True) ␣
 ↪#renaming the column of dataframe to MSE
print(loss_df) # printing the loss values corresponding to all the␣
 ↪principal components

# plotting variance plot wrt the number of principal components
fig, ax = plt.subplots()
ax.plot(loss[:,0], loss[:,1]);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='num_components', ylabel='MSE', title='MSE vs number of␣
 ↪principal components');

for l in ax.lines:
    plt.setp(l,linewidth=4)
```

```
   num_components           MSE
0             1.0  1.947004e+00
1             2.0  9.638446e-01
2             3.0  3.839144e-01
```

```
3                4.0   6.991503e-31
```



1.2.11 2.2.10 Analysis of MSE error with respect to number of principal components with non sorted eigenvalues

[45]:
```
def PCA_unsorted(X, num_components):
    """
    Args:
    X: ndarray of size (N, D), where D is the dimension of the data,
    and N is the number of datapoints
    num_components: the number of principal components to use.
    Returns:
    X_reconstruct: ndarray of the reconstruction
    of X from the first `num_components` principal components.
    """
    # your solution should take advantage of the functions you have␣
↪implemented above.
    # this function is not calculating most significant eigenvalues because␣
↪we want to use this for comparision with the earlier one
    Xbar, mu, std = normalize(X) #normalized data
    covariance = np.dot(Xbar.T,Xbar) # covariance = xbar*xbartrnaspose
    S = covariance # s is covariance
    eigvals, eigvecs = np.linalg.eig(S) # calculating eigen values and eigen␣
↪vectors (unordered)
    sum_value = sum(eigvals[:num_components])#sum of all components, we have␣
↪NOT TAKEN SORTED LIST
    B = np.stack(eigvecs[:,:num_components]) # calculating matrix B by taking␣
↪summation of num_components
    P = np.matmul(B,B.T) #P = B * Btranspose
```

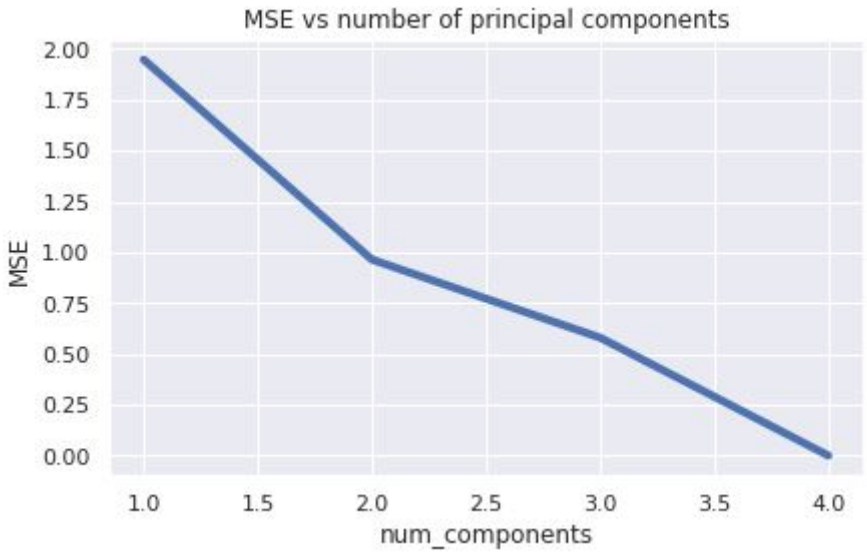
```
  X_reconstruct = np.matmul(P,X.T) #reconstructing matrix X
  X_reconstruct = X_reconstruct.T
  return X_reconstruct, sum_value, eigvals
```

[46]:
```
# defining loss, reconstructions, variance_values vector
# defining loss FOR UNSORTED DATA
loss_unsorted = []
reconstructions_unsorted = []
variance_values_unsorted = []

X1 = dataset[['rpi','spi','si','ni']]   # independent variables matrix - pollutant concentrations
Xbar1, mu1, std1 = normalize(X1)
print('Taking different values of number of components from M=1 to 4 (total number of features available): ')
# iterate over different numbers of principal components, and compute the MSE
for num_component in range(1, 5):
  reconst, sum_value, eigvals = PCA_unsorted(Xbar1, num_component)  #reconst contains reconstructed Xbar, sum value contains summation of significant of num_component
  error = mse(reconst, Xbar1)
  reconst = reconst*std1 + mu1 #reconst is un-normalized data
  reconstructions_unsorted.append(reconst) # appending un-normalized data
  print('M = {:d}, reconstruction_error = {:f}'.format(num_component,error))
  loss_unsorted.append((num_component, error)) # loss for unsorted un normalized data
  variance_values_unsorted.append((num_component, sum_value)) # variance for unsorted un normalized data

print('\nEigvalues in descending order: ')
for i in eigvals:
    print(i)
```

```
Taking different values of number of components from M=1 to 4 (total number of
features available):
M = 1, reconstruction_error = 1.947004
M = 2, reconstruction_error = 0.963845
M = 3, reconstruction_error = 0.579930
M = 4, reconstruction_error = 0.000000

Eigvalues in descending order:
894576.6285083753
428403.7878512145
167287.63150541496
252699.95213404333
```

[47]:
```
loss_unsorted = np.asarray(loss_unsorted) # converting loss to array
loss_df = pd.DataFrame(loss_unsorted) # as a dataframe
loss_df.rename(columns={0 : 'num_components', 1: 'MSE'},  inplace = True) ␣
 ↪#renaming the column of dataframe
print(loss_df)

# plotting MSE vs number of principal components
fig, ax = plt.subplots()
ax.plot(loss_unsorted[:,0], loss_unsorted[:,1]);
ax.xaxis.set_ticks(np.arange(1, 5, 5));
ax.set(xlabel='num_components', ylabel='MSE', title='MSE vs number of␣
 ↪principal components');

for l in ax.lines:
    plt.setp(l,linewidth=4)
```

	num_components	MSE
0	1.0	1.947004e+00
1	2.0	9.638446e-01
2	3.0	5.799302e-01
3	4.0	8.028734e-31



4. **Inference:**

   What you we done and why is it important?

   Dimensionality reduction is a method which helps us reduce our computation time by compressing the data-set without losing any useful information. This technique is highly used when there are more than hundreds or thousands of measurements for each data sample which often leads to failure of statistical models as well. Original Data often contains many redundant, and unimportant features that when removed improves the speed and efficiency of processing and as well as allow us to work with more compact representation of data without losing any vital information. Principal Component Analysis (PCA) is a Dimensionality Reduction Technique, which helps to visualize and compress the data. In PCA we are interested in finding projections of data points in such a way that these projections are as similar as the original data points, but which have a significantly lower intrinsic dimensionality. In our case, we first normalized the data-set and then computed the eigenvalues and eigenvectors of the corresponding features. We then sort these eigenvalues and vectors in decreasing order and then analyzed and took top M eigenvectors that greatly contribute to the data. Then, we computed the projection matrix and projected data matrix (reconstructed) that is in the same data space as the original data but with significantly lower intrinsic dimensionality. The selection of top M eigenvectors was based on the amount of variance each principal contributes and analysis of the correlation matrix which shows the collinearity between features. With PCA, our purpose is to remove the features causing multicollinearity and features that are contributing significantly less to the overall dataset. PCA is highly important as it also helps in reducing the chances of overfitting and also helps to remove unwanted features from the data which further increases the accuracy and efficiency. Applications of PCA also improves visualization of data, as it is hard to visualize data in higher dimensions. Since our dataset has more than 4 lakhs rows, high dimensions for each data point creates problems in processing the data and hence, it's highly important to reduce the dimensionality of the dataset to improve visualization and efficiency.

   Second paragraph should include how you have implemented.

   (a) Implementation:

      i. **Preprocessing of Data** : The data is pre-processed. This includes filling up null attributes with their respective averages of the column and removing the unnecessary columns of the dataset.

      ii. **Formation of feature matrix X** : The processed data set consists of 4 features that are the pollutants based on which the AQI is predicted and has 435742 samples that gives us a feature matrix of dimensions 435742 x 4.

      iii. **Normalization:** Normalization refers to shifting the distribution of each attribute to have a mean of zero and a standard deviation of one (unit variance). It is useful to standardize attributes for a model. Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like

standard normally distributed data. hence, the data is normalized before PCA is applied. as the different scales can lead to misleading components.

$$\mathbf{X} = (\mathbf{X_n} - \bar{}) / \mathbf{\text{œ}}$$

where μ is the mean of data points in X and σ is the standard deviation of X

iv. **Covariance matrix:** Now the covariance matrix is given by:

$$\mathbf{S} = \frac{1}{N}(\mathbf{X^T X})$$

where N is the total number of data points.

Covariance indicates the direction of linear relationships between variables and the matrix is a symmetric matrix.

v. **Mathematical analysis of principal components :** Now for principal components:

$$A\vec{v} = \lambda\vec{v}$$

Here A is the covariance matrix, λ is the eigen value and $\vec{v}$ is the eigen vector

$$A\vec{v} - \lambda\vec{v} = 0$$

$$\vec{v}(A - \lambda I) = 0$$

(A-λ I) has to be non-invertible

$$\det((A - \lambda I)) = 0$$

Solving the above equation gives us the set of eigen vectors and corresponding eigen-values. For python implementation, we used an inbuilt function that computes the eigen values and vectors and sorted the eigenvectors corresponding to decreasing order of eigenvalues.

vi. **Projection Matrix** : The top M (taken M =2) eigenvectors are used to make a matrix B. And further, we compute the projection matrix using:

$$P = B.B^T$$

where P = projection matrix

vii. **Projected Data Matrix ($\bar{X}$)** The projection matrix is then multiplied with the data matrix to generate new feature sub-space. This new features subspace will be such that higher variance is in more significant vectors.This will let us remove few insignificant features, thus leading to data reduction.

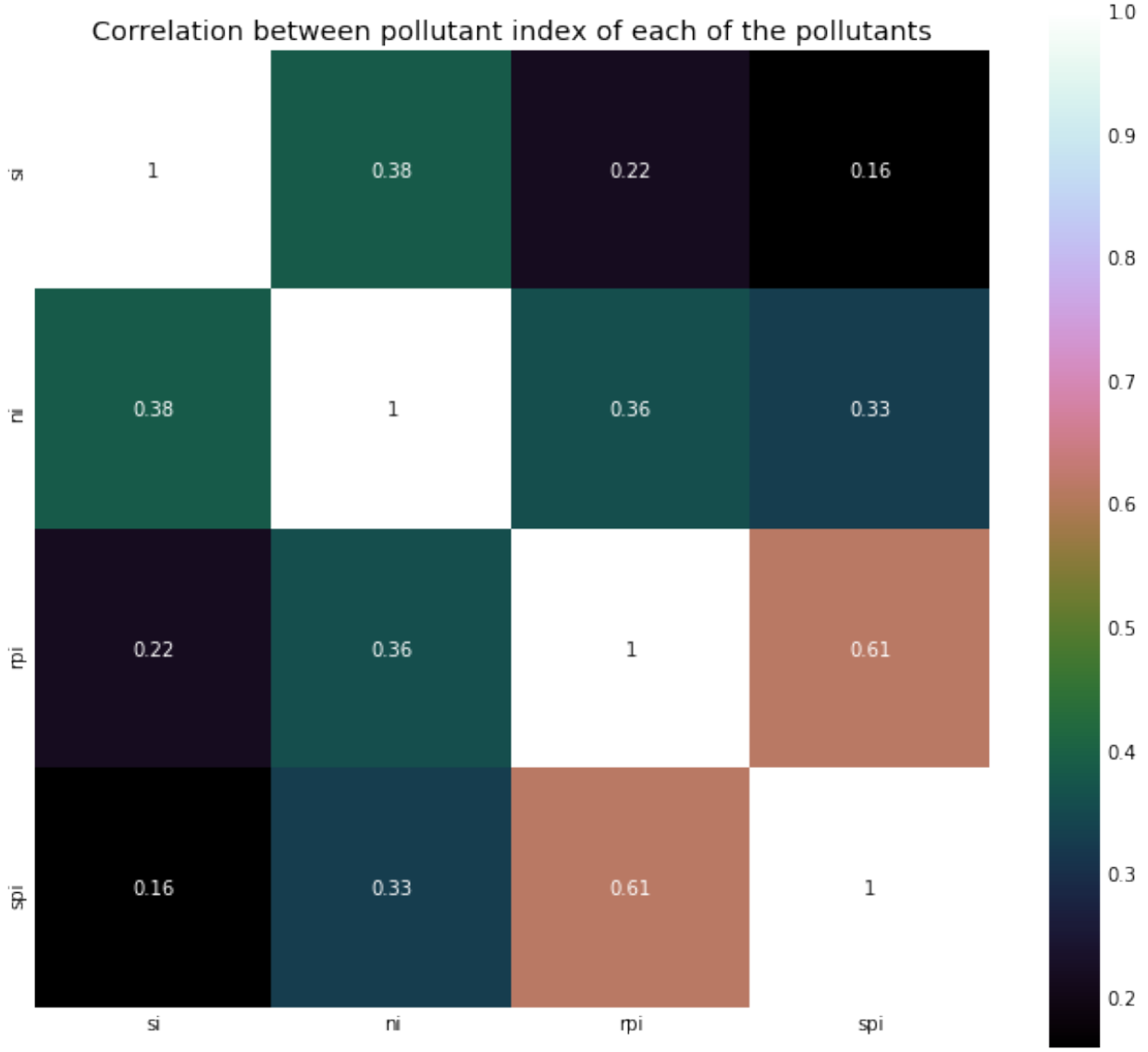
$$\bar{X} = BB^T X = PX$$

viii. **Reconstructed AQI** : From this projected data matrix which predict new AQI values and find the MSE error between Reconstructed AQI and actual AQI to analyze the effect of dimensionality reduction on the output label as well.

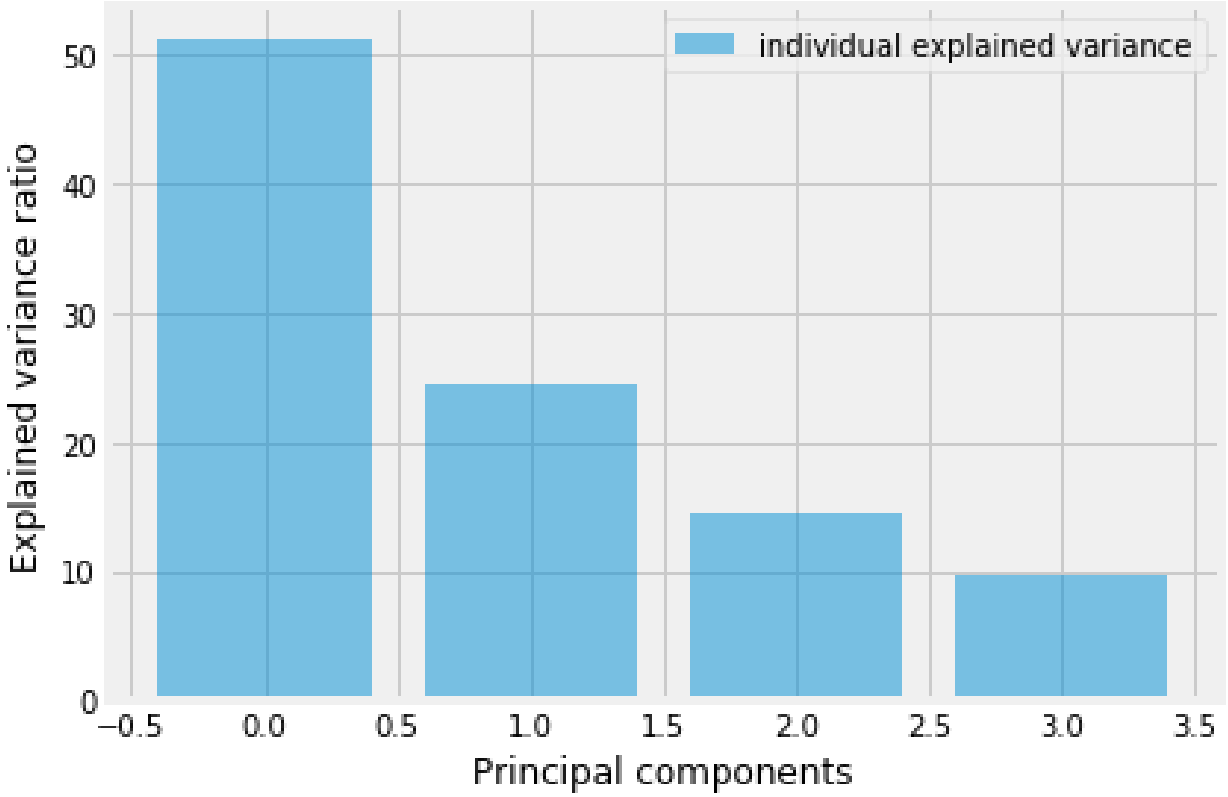
ix. **Analysis for each of the principal components** : We computed PCA considering each of the principal components and computed MSE error between the projected data matrix ($\bar{X}$) and the original matrix X for each of the princicpal components. We also compute maximum variance captured and plot it against the number of principal components.

Inferences and Results:

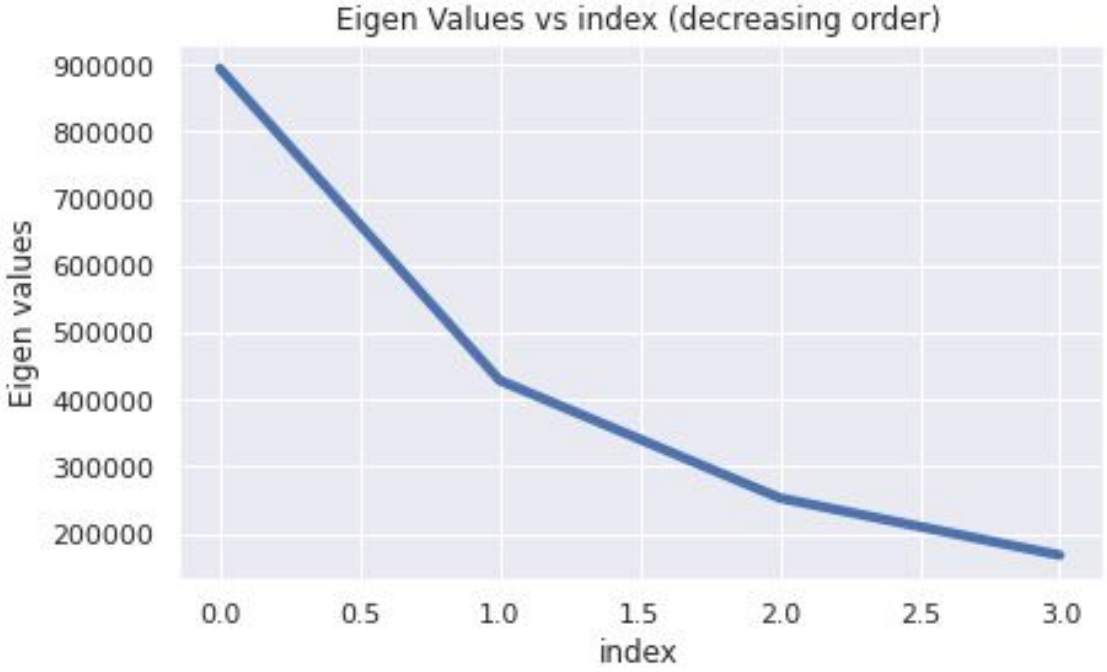
1. Correlation between the pollutant indexes The below graph shows the correlation of pollutant indexes of the pollutants - so2, no2, rpi, and spi. It compute pairwise correlation of columns.Correlation shows how the two variables are related to each other.Positive values shows as one variable increases other variable increases as well. Negative values shows as one variable increases other variable decreases.Bigger the values, more strongly two variables are correlated and vice-versa.



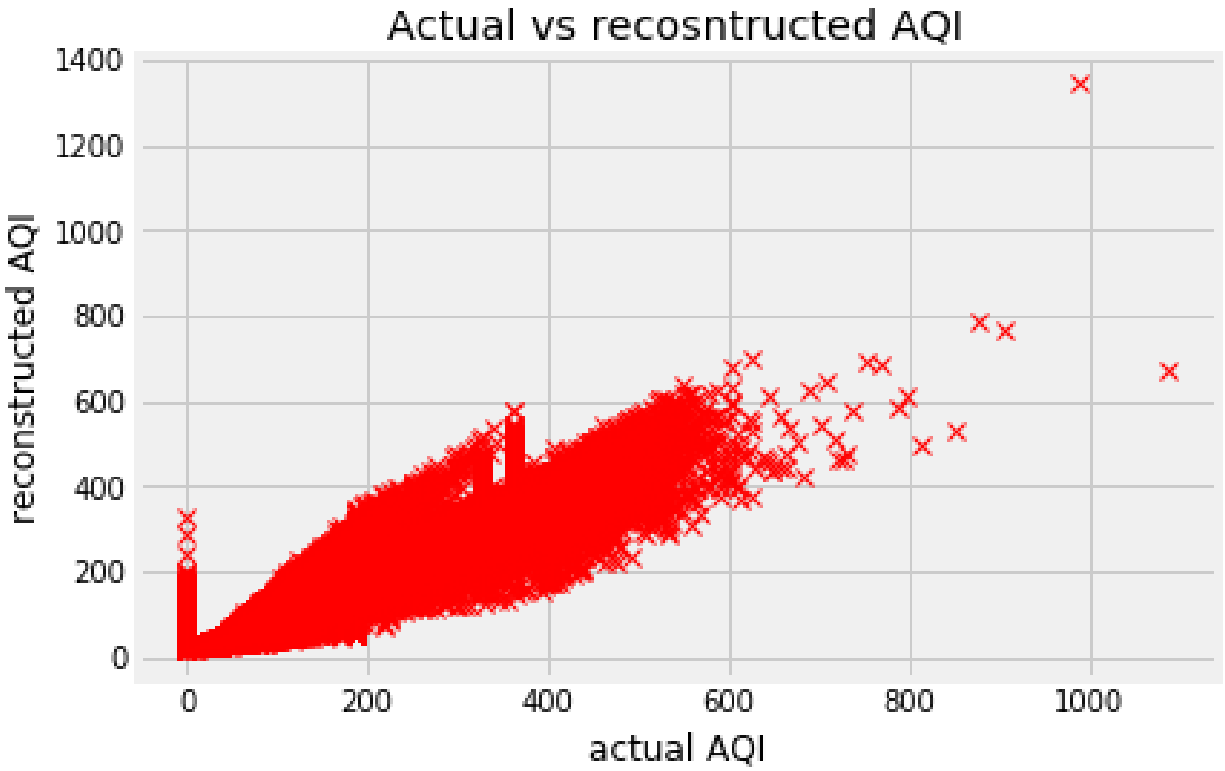
2. Explained Variance - variance attributed to each of the features After sorting the eigenpairs, the next question is "how many principal components are we going to choose for our new feature subspace?" A useful measure is the so-called "explained variance," which can be calculated from the eigenvalues. The explained variance tells us how much information (variance) can be attributed to each of the principal components. We can see from the below bar graph that 50% of the variance is contributed by the first principal component, approximately 25% of the variance is contributed by the second principal component and the rest 2 principal components contributes significantly less than than the first two components. Hence, we choose first two principal components as the low dimensional space further for analysis.



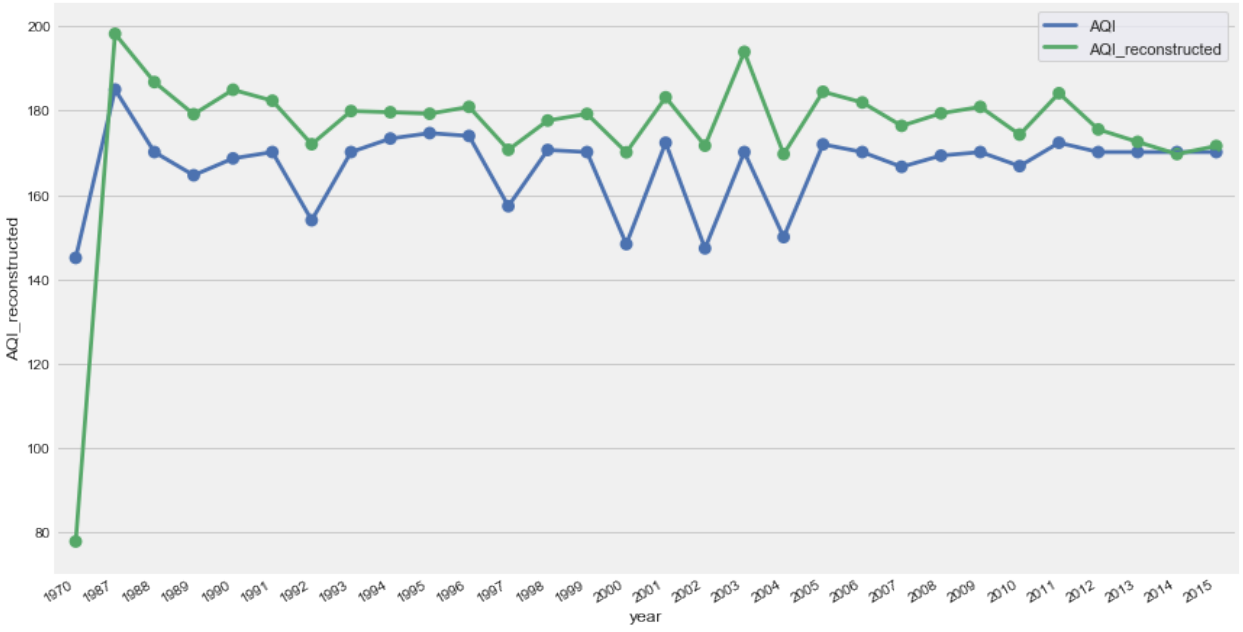
3. Sorted Eigenvalues in decreasing order The below graph shows the sorted eigenvalues in decreasing order. We have 4 features, and corresponding to that we have 4 eigenvalues and 4 corresponding eigenvectors found from the covariance matrix S. We sort the eigenvalues in decreasing order and eigenvectors correspondingly. Higher the eigenvalue, higher will be the variance contributed by that eigenvector.



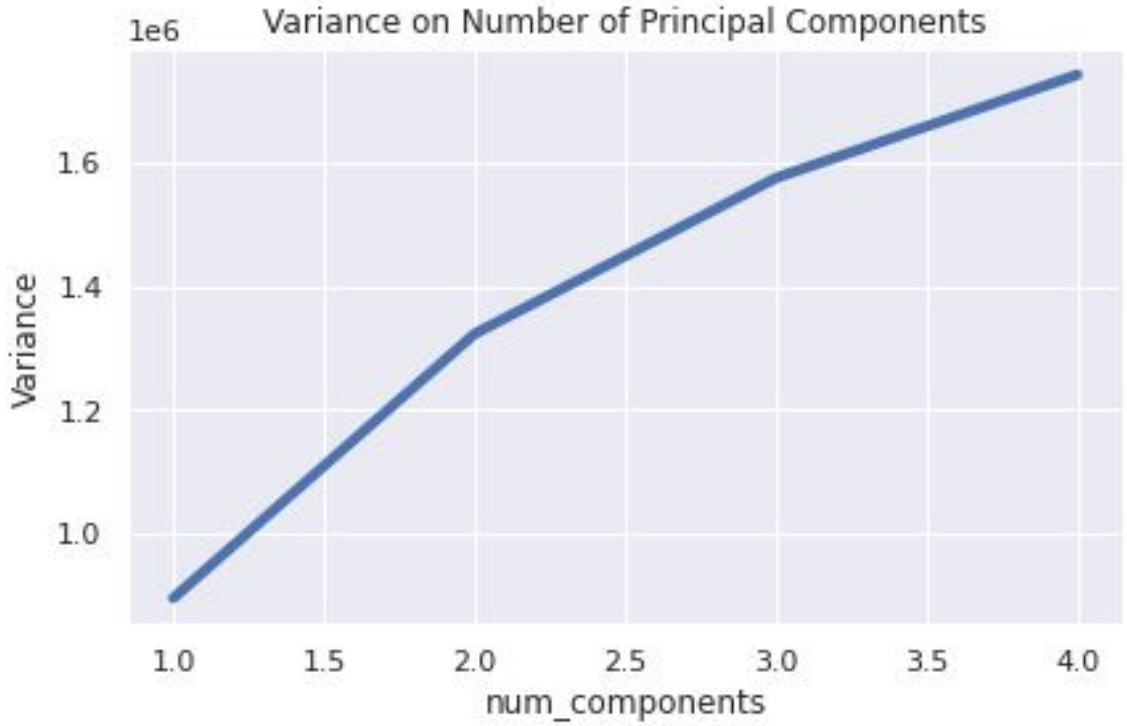
4. Actual AQI vs Reconstructed AQI from projected data matrix The below graph shows the plot between Actual AQI values and the reconstructed AQI values. The recontructed AQI values are predicted from the projected data matrix and the actual AQI values were predicted using the original input matrix X. The almost linear nature shows that the error between the actual and reconstructed AQI (from projected data matrix using 2 principal components by PCA) is less.



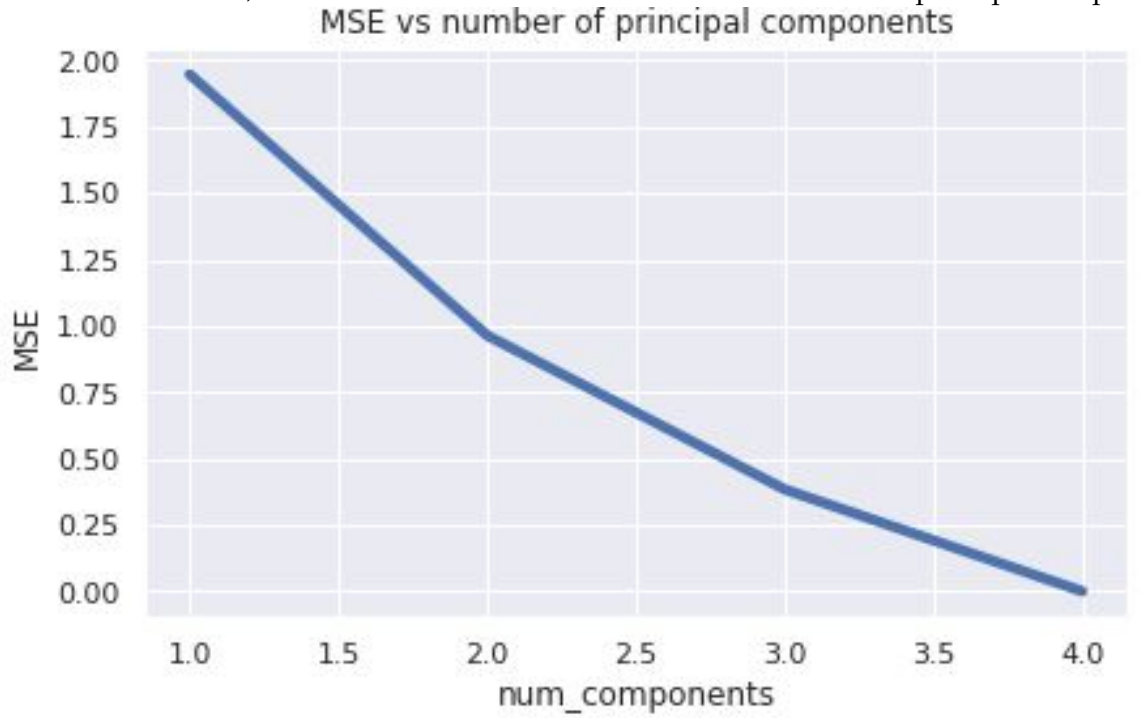
5. Actual AQI and Reconstructed AQI against year The below graph shows the Actual AQI values and reconstructed AQI values as mentioned above with respect to each year. We can clearly infer that due to loss of dimensions during PCA, the error is generated but the error between them is lesser as we have taken major principal components that highly impact the variance in the dataset and removed the less important features.



6. Maximum Captured Variance The below graph shows the maximum amount of variance with respect to the principal components. Maximum captured variance is defined by the sum of the eigenvalues of the principal components taken into consideration. Since, as the x axis represents number of principal components, it means as we go to the right, number of principal components increases and hence, variance would also increase as the number of eigenvalues in the addition is increasing. But Since, eigenvalues are sorted in decreasing order, at first the there is a rapid increase in the captured variance due to high eigenvalues, whereas as we go right, the eigenvalues that are adding have less value, hence it is then increasing though but at a slower rate.



7. MSE vs number of principal components (sorted eigenvalues) The below graph shows that the mean squared error with respect to the number of principal components. The reason is that as we take more and more principal components, there is less loss in dimensions and hence, there is less loss in data. Hence, its intuitive that error will decrease as we take more principal components.



8. MSE vs number of principal components with unsorted eigenvalues The below graph shows that the mean squared error with respect to the number of principal components. The MSE error is computed between projected data matrix and the original data. The eigen values are non ordered and not sorted before applying PCA to compute the projected data matrix and computing the MSE error. Since, there was only one change in the order of the eigenvalues and the number of features are too less, hence we can see that there is not much significant change in the nature of graph as compared to the above graph. However, if the number of features are significant large and the eigenvalues are highly unordered, then the MSE error will still decrease but will have a different shape based on the order of the eigenvalues.