```
# Importing Libraries Used.
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from sklearn.preprocessing import FunctionTransformer
from sklearn.preprocessing import PowerTransformer
from sklearn.cluster import KMeans
import seaborn as sns
from sklearn.manifold import TSNE
```

```
# Importing the data PCA
dataset = pd.read_csv('https://www.ee.iitb.ac.in/~asethi/Dump/DataClustering.csv')
```

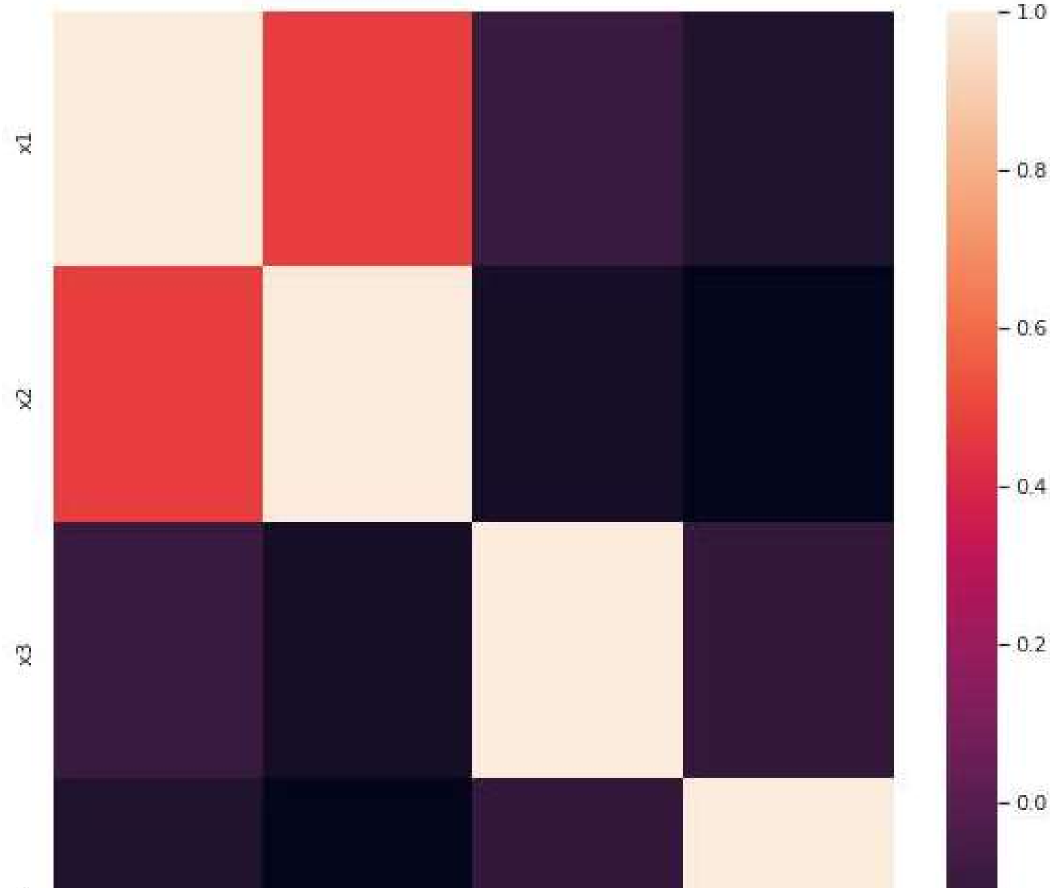
```
display(dataset)
```

	x1	x2	x3	x4
0	0.832354	1.389428	0.962226	0.993671
1	1.256087	1.500487	0.904118	0.738035
2	0.976953	1.058524	1.217530	1.357238
3	1.014365	1.122684	1.195847	0.984144
4	1.041386	1.219014	0.864819	1.720825
...	...	...	...	...
346	0.203877	0.195724	2.766999	1.826532
347	0.229380	0.131514	0.704255	2.762919
348	0.095878	0.107426	0.946789	3.434620
349	0.111690	0.130970	1.098922	2.295701
350	0.113965	0.283822	1.318260	1.494283

351 rows × 4 columns

```
# Checking Correlation between features.
corr = dataset.corr()
sns.set(rc = {'figure.figsize':(10,10)})
sns.heatmap(corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f82f80ba9d0>
```



Log Transformation

```
transformer = FunctionTransformer(np.log1p)
log_data =transformer.transform(dataset)
print(log_data)
```

```
           x1        x2        x3        x4
0    0.605602  0.871054  0.674080  0.689978
1    0.813632  0.916485  0.644019  0.552755
2    0.681557  0.721989  0.796394  0.857491
3    0.700304  0.752681  0.786568  0.685188
4    0.713629  0.797063  0.623164  1.000935
..        ...       ...       ...       ...
346  0.185547  0.178752  1.326279  1.039051
347  0.206510  0.123557  0.533128  1.325195
348  0.091556  0.102039  0.666181  1.489442
349  0.105881  0.123075  0.741424  1.192619
350  0.107925  0.249842  0.840817  0.914001

[351 rows x 4 columns]
```

Power Transformation

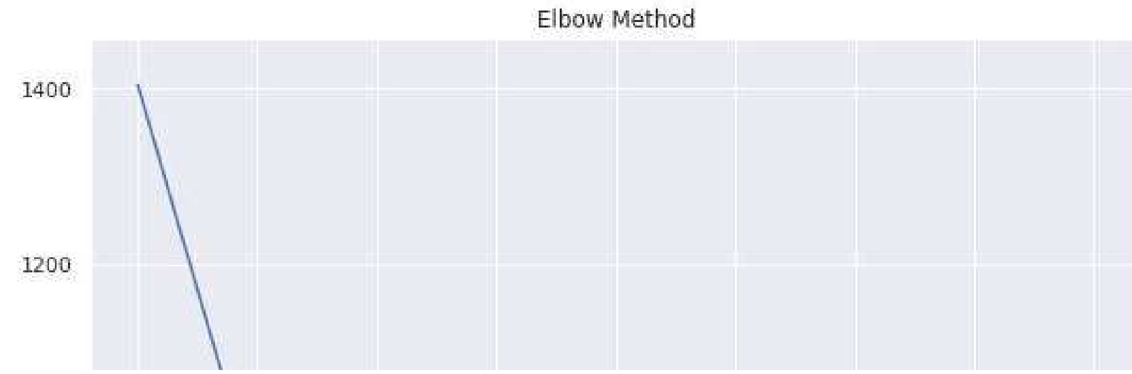
```
pt = PowerTransformer()  # yeo-johnson method (default)
power_data = pt.fit_transform(dataset)
print(power_data)
print(pt.lambdas_)
```

```
[[ 1.89289283  1.52485355 -0.34007498 -0.41407712]
 [ 1.98737654  1.56223077 -0.47399533 -1.03977999]
 [ 1.94153709  1.36253967  0.13915026  0.21840941]
 ...
 [-0.94716647 -0.87513322 -0.37459819  1.70555679]
 [-0.70003814 -0.71928565 -0.0638882   1.14947884]
 [-0.66648912  0.04746441  0.28985554  0.40376835]]
[-6.06020316 -2.81395054 -1.7083417  -1.24217325]
```

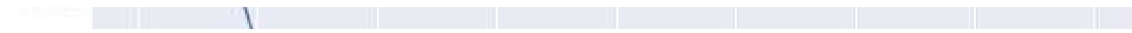
Elbow Method for FInding optimal K value

For power transformed data--

```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 10):
    kmeans = KMeans(n_clusters = i, init = 'k-means++')
    kmeans.fit(power_data)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 10), wcss)
plt.title('Elbow Method')
plt.xlabel('No. of clusters')
plt.ylabel('wcss')
plt.show()
```



For log Transformed data---

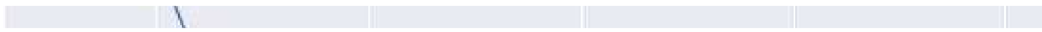


```
from sklearn.cluster import KMeans
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++')
    kmeans.fit(log_data)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('No. of clusters')
plt.ylabel('wcss')
plt.show()
```

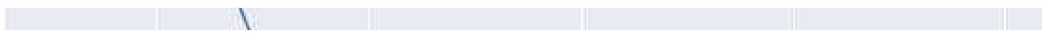
Observations

We can see optimal value of K comes out be K=4 when we apply Log transformation.

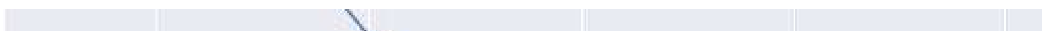
Training Kmeans



```
k_means = KMeans(n_clusters = 4, init = 'k-means++')
y_k_means = k_means.fit_predict(log_data)
```

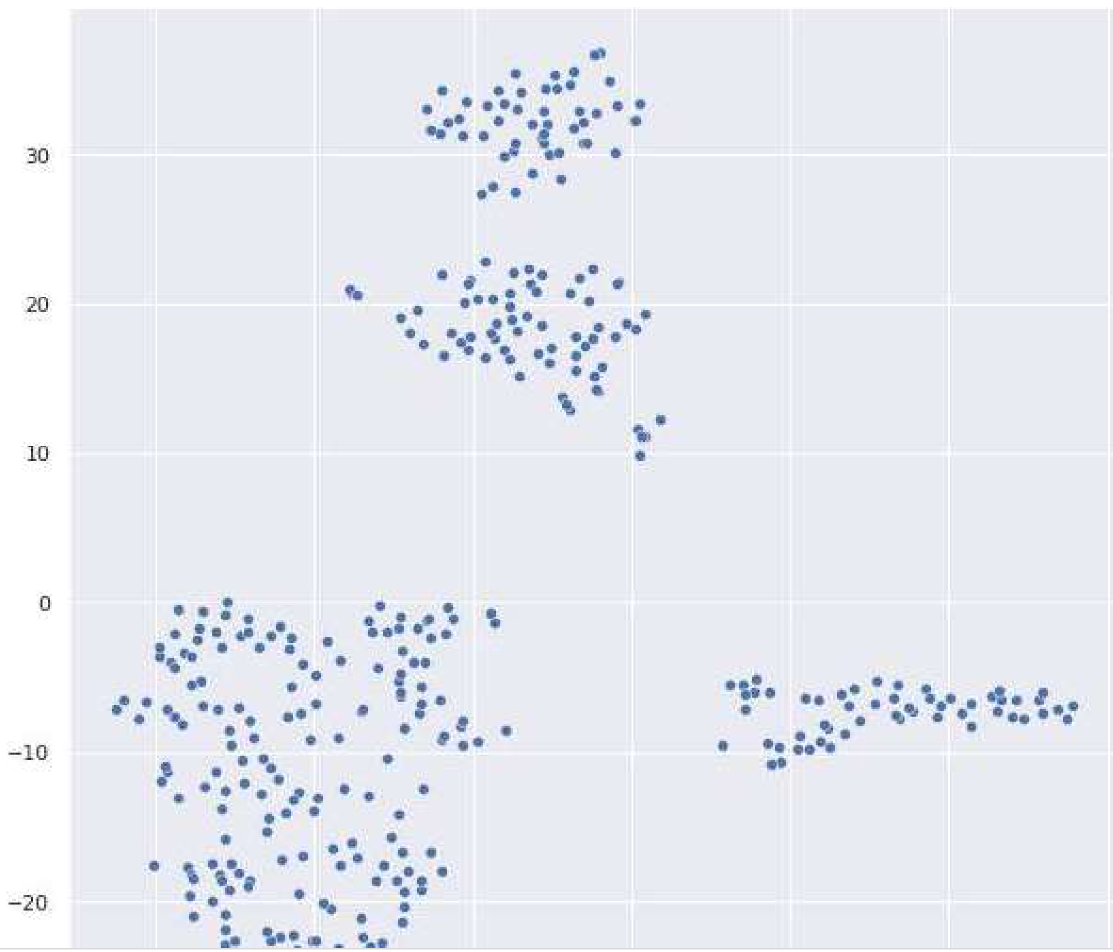


Visulazise clusters



```
tsne = TSNE(n_components=2, verbose=1, random_state=123)
tsne_data = tsne.fit_transform(log_data)
sns.scatterplot(x= tsne_data[:,0], y= tsne_data[:, 1])
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:783: FutureWarning
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/manifold/_t_sne.py:793: FutureWarning
  FutureWarning,
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 351 samples in 0.001s...
[t-SNE] Computed neighbors for 351 samples in 0.012s...
[t-SNE] Computed conditional probabilities for sample 351 / 351
[t-SNE] Mean sigma: 0.170476
[t-SNE] KL divergence after 250 iterations with early exaggeration: 56.775246
[t-SNE] KL divergence after 1000 iterations: 0.316018
<matplotlib.axes._subplots.AxesSubplot at 0x7f82f2a34f10>
```



EE769 Assignment 3 Part 3 , (PCA)

```
# Importing Libraries Used.
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.metrics import mean_squared_error
```

```
# Importing the data PCA
dataset = pd.read_csv('https://www.ee.iitb.ac.in/~asethi/Dump/DataPCA.csv')
```

```
display(dataset)
```

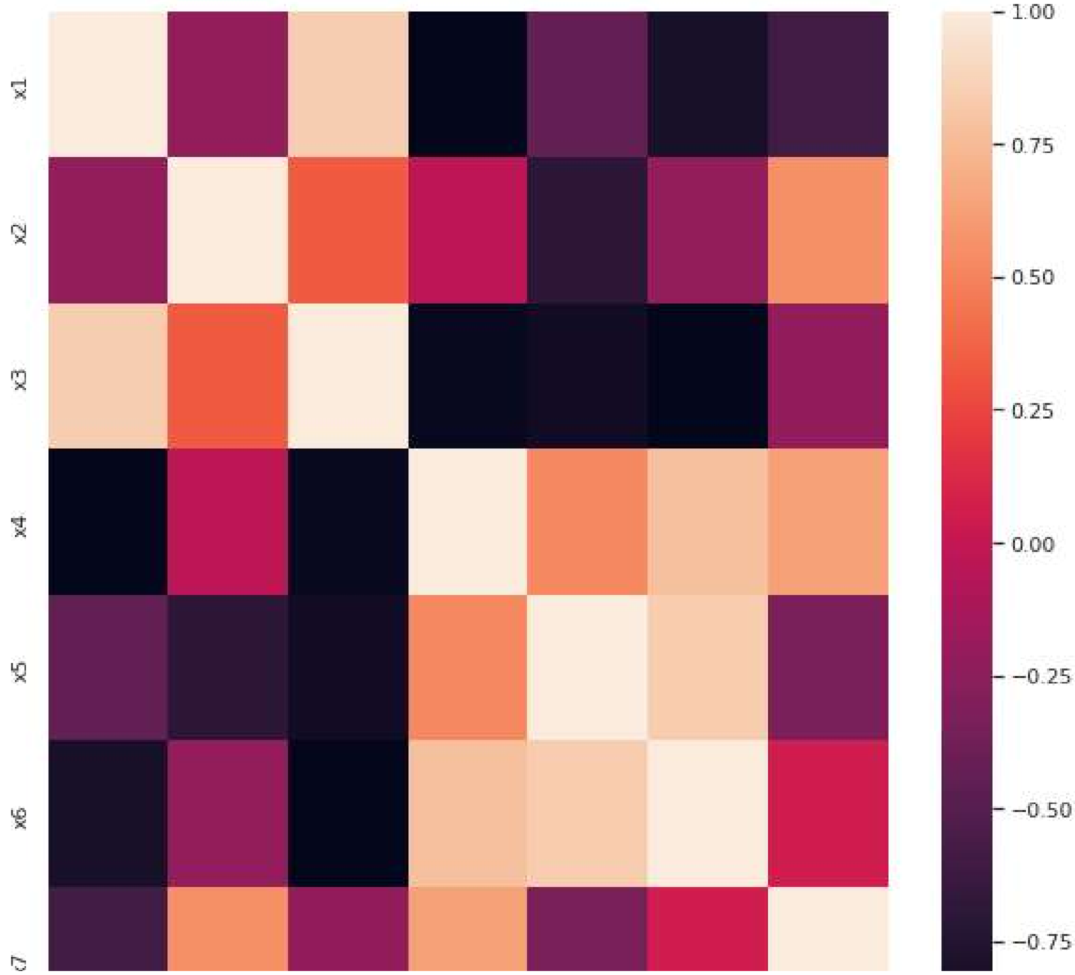
	x1	x2	x3	x4	x5	x6	x7
0	0.840261	-1.088160	4.861744	4.273055	4.312457	-0.137834	0.076453
1	1.320591	-1.174113	5.247360	3.295027	4.283410	-0.363759	-0.170605
2	1.537909	-1.175882	5.556251	3.394183	3.971574	-0.888398	0.080617
3	0.363552	-1.130608	4.329890	5.547488	4.539732	0.342330	0.251953
4	1.567938	-1.114719	5.542104	2.493071	4.156157	-0.609694	-0.291367
...	...	...	...	...	...	...	...
185	0.894296	-1.270097	4.618647	4.005702	4.634847	0.156118	-0.215169
186	0.926559	-1.203508	4.766199	3.907673	4.538084	0.073321	-0.197924
187	1.678706	-1.227245	5.613809	2.864530	4.046335	-0.906304	-0.099063
188	1.693254	-1.323340	5.470576	3.023584	4.208331	-0.693895	-0.189082
189	1.806170	-1.347104	5.541666	2.818543	4.240505	-0.690840	-0.222595

190 rows × 7 columns

Visuilizing Data

```
# Checking Correlation between features.
corr = dataset.corr()
sns.set(rc = {'figure.figsize':(10,10)})
sns.heatmap(corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6a9c36d250>
```




```
# Feature Scaling applied to given data before implementing PCA.
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
features = sc.fit_transform(dataset)


print(features)
print(features.size)
```

```
[[-0.87970185  0.20358106 -0.7178119  ...  0.24030982  0.572178
   0.81782988]
 [ 0.06690112 -0.16483345 -0.01377784 ...  0.15160586  0.10228252
  -0.215624  ]
 [ 0.49517799 -0.17241562  0.55017623 ... -0.80067599 -0.98890001
   0.83524673]
 ...
 [ 0.7726511  -0.39257257  0.65526181 ... -0.57237401 -1.02614162
   0.08363971]
 [ 0.80132199 -0.80445602  0.39375587 ... -0.07767042 -0.58435868
  -0.29291356]
 [ 1.02385053 -0.906317    0.5235481  ...  0.02058323 -0.57800454
  -0.43309953]]
1330
```

```
# Implementing PCA
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = None)
features_new = pca.fit_transform(features)
print(features_new)
print(features_new.shape)
```

```
[[-1.59702425e+00  8.26765877e-01 -1.68860477e-01 ... -3.81497945e-04
  -1.25234401e-02  9.67038931e-04]
 [-4.47094849e-02 -3.26854584e-01  5.90225928e-02 ...  9.35536010e-03
   1.51737615e-02 -1.46186620e-02]
 [ 1.05989206e+00  7.42774576e-01 -1.10013535e+00 ...  4.22191049e-02
   4.61237385e-04 -6.96580057e-03]
 ...
 [ 1.49069416e+00 -1.00477599e-01 -7.51700799e-01 ...  1.70219952e-02
   1.24586688e-02  2.34403848e-04]
 [ 9.32145740e-01 -8.09761124e-01 -6.65205848e-01 ...  5.32175046e-02
  -5.65380029e-03  8.28166020e-04]
 [ 1.15386944e+00 -1.08824752e+00 -6.24044099e-01 ... -6.16341540e-02
  -4.32827044e-03 -7.63587531e-03]]
(190, 7)
```

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

```
[6.09270567e-01 3.19460967e-01 6.49562011e-02 6.07238580e-03
 2.16027661e-04 1.98460843e-05 4.00494511e-06]
```

Observations

PC1= 60.92% , PC2 = 31.946% , PC3 = 6.495% , PC4 = 0.6072% , PC5 = 0.02160% , PC6 = 1.984e-03% , PC7 = 4.004e-04%

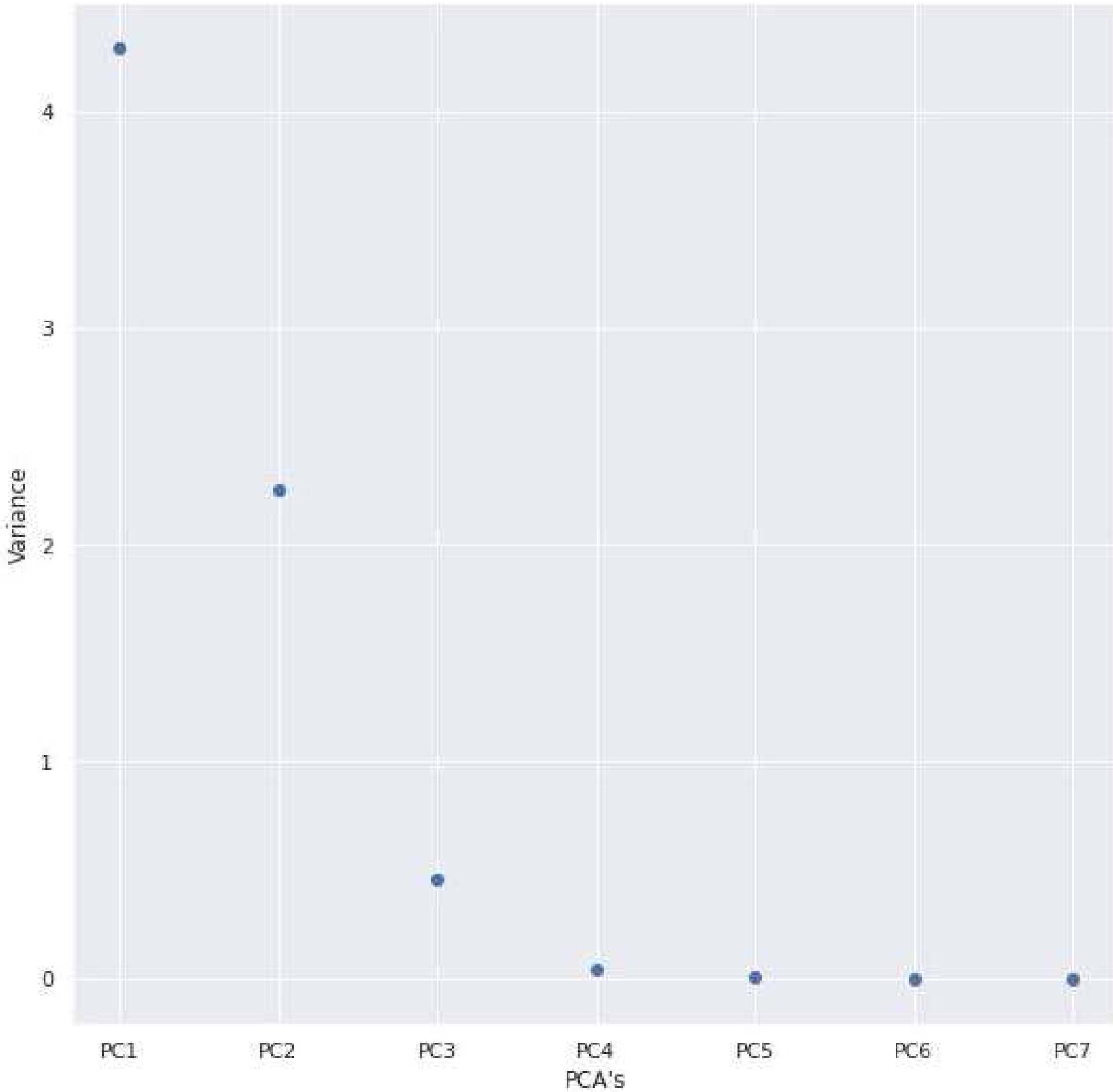
```
# Variance Explained
explained_variance =pca.explained_variance_
print(explained_variance)
```

```
[4.28745955e+00 2.24805866e+00 4.57099193e-01 4.27316037e-02
 1.52019465e-03 1.39657631e-04 2.81829470e-05]
```

Plotting Variance explained vs PCA dimensions.

```
# Plotting Variance explained vs PCA dimensions.
PCA_axis =['PC1' , 'PC2' , 'PC3' , 'PC4', 'PC5' , 'PC6' , 'PC7']
plt.scatter(PCA_axis , explained_variance)
plt.xlabel("PCA's")
plt.ylabel("Variance")
plt.show
```

```
<function matplotlib.pyplot.show>
```



Observations

---

1. We can see that PC1 , PC2 , PC3 consists of 99.361%. So we will use just PC1 , PC2 and PC3 for reconstruction.

Reconstructing the data with 3 PCA dimensions

Using only 3 PCA dimensions

```
pca = PCA(n_components = 3)
features_new_dim_red = pca.fit_transform(features)
print(features_new_dim_red)
```

```
[[-1.59702425e+00  8.26765877e-01 -1.68860477e-01]
 [-4.47094849e-02 -3.26854584e-01  5.90225928e-02]
 [ 1.05989206e+00  7.42774576e-01 -1.10013535e+00]
```

```
 [-3.88950822e+00  1.21477957e+00 -4.30487010e-01]
 [ 1.26840455e+00 -5.23475258e-01  4.06015894e-01]
 [ 3.77951767e+00 -1.90569443e-01  4.54098211e-01]
 [-3.00701913e+00 -1.33502875e+00 -8.94411173e-01]
 [-2.62335510e+00 -1.87596181e+00  1.01301160e+00]
 [ 2.82029180e-01  2.88133953e-01 -1.06697563e-01]
 [-3.33413162e-01  3.38759789e-02 -4.29663720e-01]
 [ 4.83092767e+00 -6.12540389e-01 -5.48718159e-01]
 [ 3.84008060e+00  5.54928931e-01  5.46026892e-01]
 [-2.53557460e+00  5.59603395e-01  6.54030041e-01]
 [ 3.48050273e+00 -7.93793650e-01  6.10516003e-01]
 [-2.20708361e+00 -2.46790988e-01 -1.30506506e+00]
 [-2.70175173e+00  1.90107275e-01 -4.14335816e-01]
 [ 4.15230351e-02  1.06500780e+00 -6.26336039e-01]
 [ 1.45834397e+00 -2.21829034e+00 -6.82418245e-02]
 [-4.83915879e-01  1.07920891e+00  9.77570078e-01]
 [-5.52503929e-01 -1.54533457e+00  3.41383106e-01]
 [-3.28637532e-01 -1.39366315e+00  1.09743949e-01]
 [ 2.38109312e+00  2.06548265e-01 -1.04785253e+00]
 [ 3.74543219e-01  1.45074615e-01 -1.21231325e+00]
 [-1.74522526e+00 -4.08957271e-02  9.96817256e-01]
 [-3.92685339e+00 -1.74380018e+00  1.41733331e+00]
 [-3.26075740e+00 -5.02107042e-01  2.17478661e-01]
 [ 8.83896926e-02 -2.51641426e-01 -5.20828844e-01]
 [ 5.93690909e+00  1.32610199e+00 -8.63972982e-01]
 [-2.98656926e+00  3.15860077e+00  2.88172474e-01]
 [ 1.65798237e+00  3.94840887e-01 -1.16373751e-01]
 [ 8.99905812e-02  2.05347761e-01 -1.48081174e-01]
 [ 1.17320524e+00 -2.07006948e-01 -4.92490358e-01]
 [-6.36551376e-01  1.79942980e+00 -4.37044738e-01]
 [ 8.22130885e-02  5.20647789e-01 -3.66578561e-01]
 [-2.09233969e+00 -1.06004323e+00 -2.09818004e-01]
 [ 4.79486393e-01 -7.71327072e-01  2.26165789e-01]
 [ 3.67531864e-01 -8.25482855e-01 -1.04829219e+00]
 [ 1.73442765e+00 -8.21205868e-01 -7.65461012e-01]
 [ 2.99493094e+00  1.25918585e-01  6.45633466e-02]
 [ 2.16781914e+00 -9.21758451e-01  5.14049707e-01]
 [-4.69447557e-02  3.22546162e+00  1.17281739e+00]
 [ 9.83230627e-01  3.31319458e+00  9.00355670e-01]
 [-1.47218621e+00 -1.09558961e+00 -8.35252843e-01]
 [ 1.57366263e+00  8.16089713e-02  3.83217107e-01]
 [ 2.39544781e+00  6.75791173e-01  5.93868604e-01]
 [-1.20588191e+00 -3.15791221e-01  6.37876790e-02]
 [-9.30731801e-02 -4.42671964e-01 -9.81564039e-01]
 [-1.01969437e+00 -7.17408344e-01 -5.88358042e-02]
 [-3.77115238e+00  7.86163172e-01 -3.39465081e-01]
 [-6.94551949e-01  4.68951868e-01 -1.10941295e-01]
 [ 2.10917283e+00  9.79980376e-01 -3.32946511e-02]
 [ 1.41014686e+00  1.99097934e+00 -2.60972881e-01]
 [-2.14215569e+00  1.75873234e+00 -2.82243436e-01]
 [ 1.88302180e+00 -6.95305795e-01 -6.31996135e-03]
 [ 1.46822079e+00  1.05898358e+00 -1.39354635e+00]
 [-7.92818715e-01  2.79644855e-01 -2.19742148e-01]
 [-3.81218991e-01 -2.26552253e+00  4.03825369e-01]
 [-3.07572353e-01  4.40281870e+00 -1.20597780e-01]
```

```
transformed_original_data = pca.inverse_transform(features_new_dim_red )
transformed_original_data_std = sc.inverse_transform(transformed_original_data)
```

```
print(transformed_original_data_std)
```

```
[[ 0.8342799  -1.08870918  4.86475515 ...  4.31345808 -0.14069837
   0.07595683]
 [ 1.31757016 -1.17224852  5.23163884 ...  4.28539786 -0.37489712
  -0.17090838]
 [ 1.51340028 -1.1751035   5.53124667 ...  3.97873188 -0.91385259
   0.0824435 ]
 ...
 [ 1.6956865  -1.22613991  5.62008513 ...  4.04863809 -0.89295309
  -0.09494161]
 [ 1.665073   -1.32336178  5.44736893 ...  4.21669957 -0.71869633
  -0.18578369]
 [ 1.74934756 -1.34944424  5.4954131  ...  4.22849558 -0.74484153
  -0.23818417]]
```

Calculating Mean Square Error

```
mse = (np.sum((transformed_original_data-features)**2)) / transformed_original_data_std.si
#mse1 = mean_squared_error(features, transformed_original_data ,squared=True)
print(mse)
#print(mse1)
```

```
0.006312264485441588
```

MSE values for various number of PCA's Dimensions used.

```
for i in range(1,8):

  pca = PCA(n_components = i)
  features_new_dim_red = pca.fit_transform(features)
  #print(features_new_dim_red)
  transformed_original_data = pca.inverse_transform(features_new_dim_red )
  transformed_original_data_std = sc.inverse_transform(transformed_original_data)
  #print(transformed_original_data_std)
  mse = (np.sum((transformed_original_data-features)**2)) / transformed_original_data.size
  print(mse)
```

```
0.39072943283022843
0.07126846555354302
0.006312264485441588
0.00023987869019137428
2.3851029452934055e-05
4.004945107075588e-06
9.911601657016492e-31
```

Observations

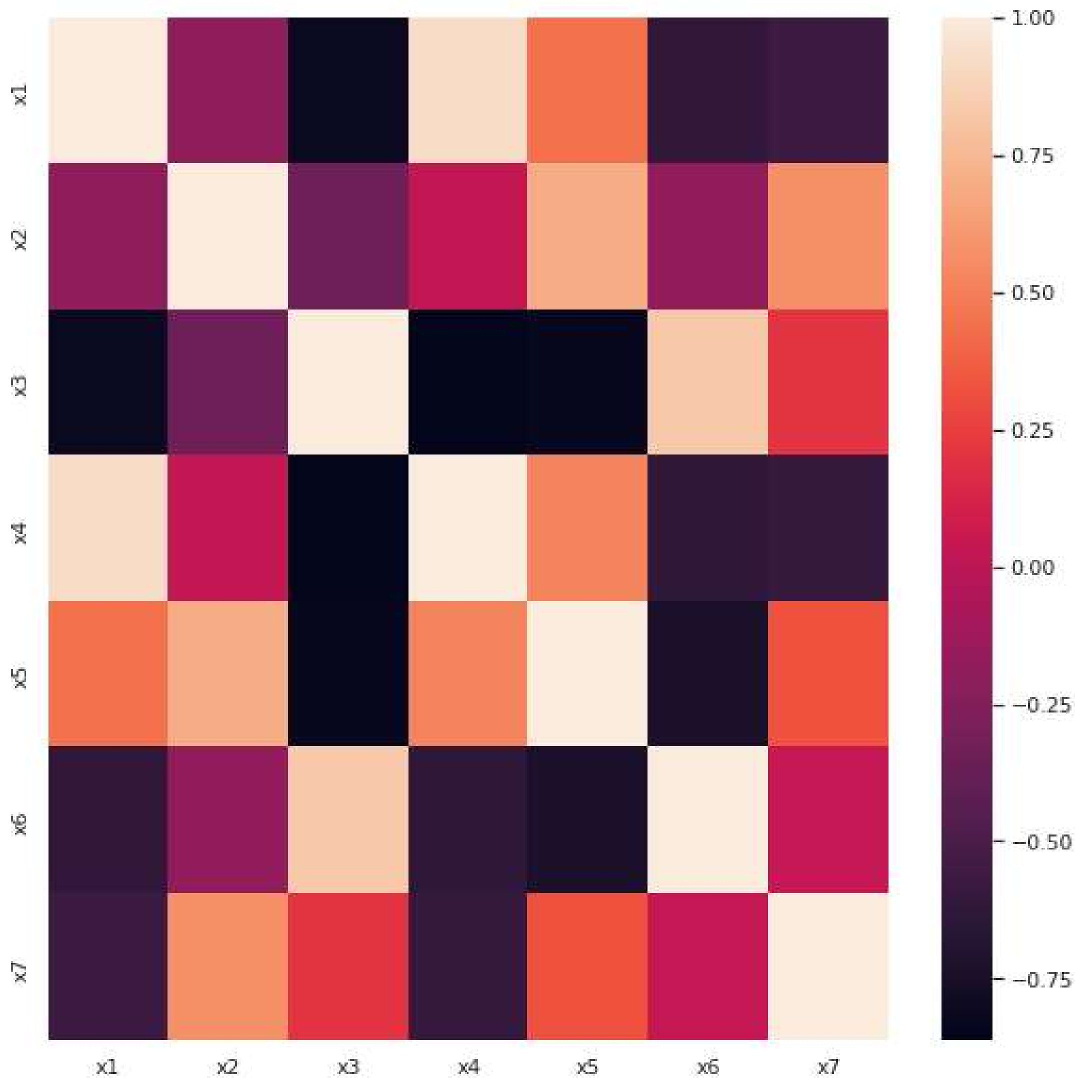
1. The mean square error between the original data and the data obtained by inverse transformation of 1 PCA componets is 0.390.
2. The mean square error between the original data and the data obtained by inverse transformation of 2 PCA componets is 0.071.
3. The mean square error between the original data and the data obtained by inverse transformation of 3 PCA componets is 0.00631.
4. The mean square error between the original data and the data obtained by inverse transformation of 4 PCA componets is 0.0002398.
5. The mean square error between the original data and the data obtained by inverse transformation of 5 PCA componets is 2.3851e-05.
6. The mean square error between the original data and the data obtained by inverse transformation of 6 PCA componets is 4.0049e-06.
7. The mean square error between the original data and the data obtained by inverse transformation of 7 PCA componets is 9.91160e-31.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
```

```
# Importing the data PCA
dataset = pd.read_csv('https://www.ee.iitb.ac.in/~asethi/Dump/DataKPCA.csv')
```

```
# Checking Correlation between features.
corr = dataset.corr()
sns.set(rc = {'figure.figsize':(10,10)})
sns.heatmap(corr)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7ff0d911c710>




```
# Feature Scaling applied to given data before implementing PCA.
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
features = sc.fit_transform(dataset)
```

```
print(features)
features.shape
```

```
[[-0.89353091  0.21422802  0.66042339 ... -0.1801732   0.26302041
   0.75031082]
 [-0.04301419 -0.15508087 -0.10846747 ... -0.08927182 -0.26849909
  -0.33491828]
 [ 0.40630971 -0.16271138 -0.63691522 ...  0.82338803 -0.88452044
   0.77101109]
 ...
 [ 0.71887615 -0.38479911 -0.72678584 ...  0.61512483 -0.8902921
  -0.04960267]
 [ 0.75213495 -0.80302959 -0.49813955 ...  0.14103532 -0.75695819
  -0.40477201]
 [ 1.01639752 -0.90700998 -0.61371353 ...  0.04316081 -0.75400694
  -0.52745531]]
(190, 7)
```

```
# Implementing KPCA
from sklearn.decomposition import KernelPCA
kpca = KernelPCA(kernel='rbf', n_components= None)
features1 = kpca.fit_transform(features)
print(features1)
print(features1.shape)
```

```
[[-5.11020702e-01  2.23517756e-01 -1.97079133e-01 ... -2.55600247e-05
   2.11452755e-05  4.69027574e-07]
 [ 5.38125267e-03 -2.35126045e-01 -4.85854890e-01 ... -1.06651680e-04
   2.06436705e-05 -4.08931280e-05]
 [ 2.55309779e-01  1.97993115e-01 -3.38920048e-01 ... -1.00005527e-05
  -1.47164996e-05 -9.99370393e-06]
 ...
 [ 4.88125969e-01 -1.04044323e-01 -2.34559167e-01 ...  7.95875487e-05
   1.08474576e-05  3.85601208e-05]
 [ 3.56596771e-01 -3.57040072e-01 -2.33242369e-01 ...  7.63076705e-05
   1.74041438e-05 -1.22113066e-04]
 [ 4.20115979e-01 -3.93452796e-01 -1.06936366e-01 ...  3.10693424e-05
  -2.71487234e-05  2.77918209e-05]]
(190, 189)
```

Calculation of Varinace

```
eigen_values = kpca.eigenvalues_
print(eigen_values)
```

```
[2.81350179e+01 2.01993011e+01 1.89193362e+01 8.70164712e+00
 8.10980162e+00 6.51442556e+00 5.53496135e+00 3.88807189e+00
 3.15337111e+00 2.72547267e+00 2.61678160e+00 2.30269553e+00
 2.00235830e+00 1.67500575e+00 1.47385909e+00 1.20272465e+00
 1.14534882e+00 9.57227423e-01 8.41495328e-01 7.66123203e-01
 7.29514608e-01 6.89916661e-01 6.24477105e-01 6.14626408e-01
 5.87489028e-01 4.57013363e-01 4.15499045e-01 3.76033025e-01
 3.45300338e-01 3.14633798e-01 2.74594879e-01 2.61967648e-01
 2.41535510e-01 2.12102700e-01 1.96125944e-01 1.89868476e-01
```

```
 1.75414567e-01 1.68382580e-01 1.58029646e-01 1.36311669e-01
 1.27258988e-01 1.23356493e-01 1.10171137e-01 1.01088155e-01
 9.41703383e-02 8.71359336e-02 8.08498068e-02 7.27737466e-02
 6.92917601e-02 6.56270812e-02 5.89121825e-02 5.68698015e-02
 5.40117666e-02 4.95978651e-02 4.84960599e-02 4.57033276e-02
 4.28721664e-02 3.80672372e-02 3.68922891e-02 3.15981637e-02
 3.11592228e-02 2.96456557e-02 2.78825746e-02 2.48852016e-02
 2.32687022e-02 2.21642300e-02 2.03821843e-02 1.89597746e-02
 1.73823022e-02 1.60490456e-02 1.57897224e-02 1.50966864e-02
 1.32688343e-02 1.26178960e-02 1.13339090e-02 1.07541458e-02
 1.02074462e-02 9.78797111e-03 8.70180548e-03 8.36568077e-03
 7.65546011e-03 7.15712992e-03 6.99731811e-03 6.43102679e-03
 6.02639884e-03 5.36356290e-03 4.98694571e-03 4.65718732e-03
 4.54301228e-03 4.42622238e-03 4.32121005e-03 3.87947031e-03
 3.59384291e-03 3.48499390e-03 3.26940253e-03 3.16787640e-03
 2.87181830e-03 2.85661089e-03 2.58401084e-03 2.42790168e-03
 2.26595027e-03 2.05070909e-03 1.98191066e-03 1.90835703e-03
 1.86725434e-03 1.70275036e-03 1.59351746e-03 1.47268838e-03
 1.41225051e-03 1.27845170e-03 1.22585032e-03 1.15882395e-03
 1.11861455e-03 1.06882420e-03 9.96134673e-04 9.75017681e-04
 9.30410456e-04 8.05641669e-04 7.96043095e-04 7.12284285e-04
 6.68695595e-04 6.14707903e-04 5.66925966e-04 5.48005653e-04
 5.21117356e-04 4.91293876e-04 4.58357631e-04 4.35927184e-04
 4.05151822e-04 3.72790602e-04 3.25781471e-04 3.09523433e-04
 3.02032552e-04 2.78462053e-04 2.66759763e-04 2.53306359e-04
 2.31835209e-04 2.26628542e-04 2.09128933e-04 1.82349523e-04
 1.79223969e-04 1.65628763e-04 1.57625237e-04 1.48303827e-04
 1.31306623e-04 1.22637607e-04 1.12678212e-04 1.05336856e-04
 1.00475272e-04 9.39319509e-05 8.93933919e-05 8.21826301e-05
 7.74882841e-05 7.57386174e-05 6.78973414e-05 6.25569113e-05
 5.71453338e-05 5.64136292e-05 5.29866018e-05 4.74850019e-05
 4.23326412e-05 4.08495147e-05 3.59056492e-05 3.21320882e-05
 3.02343500e-05 2.64280748e-05 2.50306333e-05 2.20133519e-05
 2.06557030e-05 1.99846699e-05 1.85986845e-05 1.52526327e-05
 1.25776366e-05 1.23140093e-05 1.06050255e-05 1.03940765e-05
 8.21565158e-06 7.15194197e-06 6.73373928e-06 6.06323324e-06
 5.20049557e-06 4.17288389e-06 3.55859357e-06 2.51356171e-06
 2.49662589e-06 2.02910774e-06 1.55907365e-06 1.04347408e-06
 7.64583978e-07]
```

```
var = eigen_values/sum(eigen_values)
print(var)
```

```
[2.16582460e-01 1.55493568e-01 1.45640440e-01 6.69849988e-02
 6.24289913e-02 5.01478379e-02 4.26079540e-02 2.99302520e-02
 2.42745491e-02 2.09806007e-02 2.01439004e-02 1.77260759e-02
 1.54140896e-02 1.28941402e-02 1.13457197e-02 9.25853554e-03
 8.81685830e-03 7.36870582e-03 6.47780389e-03 5.89759170e-03
 5.61577992e-03 5.31095621e-03 4.80720461e-03 4.73137426e-03
 4.52247159e-03 3.51807413e-03 3.19849825e-03 2.89469010e-03
 2.65811087e-03 2.42204083e-03 2.11382252e-03 2.01661851e-03
 1.85933257e-03 1.63275974e-03 1.50977119e-03 1.46160140e-03
 1.35033568e-03 1.29620367e-03 1.21650712e-03 1.04932283e-03
 9.79635588e-04 9.49594305e-04 8.48093859e-04 7.78173356e-04
 7.24920227e-04 6.70769606e-04 6.22379205e-04 5.60209954e-04
 5.33405735e-04 5.05195154e-04 4.53504081e-04 4.37781897e-04
 4.15780837e-04 3.81802766e-04 3.73321104e-04 3.51822742e-04
 3.30028554e-04 2.93040364e-04 2.83995651e-04 2.43241645e-04
 2.39862692e-04 2.28211301e-04 2.14639160e-04 1.91565479e-04
```

```
 1.79121719e-04 1.70619528e-04 1.56901397e-04 1.45951733e-04
 1.33808402e-04 1.23545035e-04 1.21548774e-04 1.16213805e-04
 1.02143058e-04 9.71321567e-05 8.72480660e-05 8.27850673e-05
 7.85765916e-05 7.53474859e-05 6.69862179e-05 6.43987407e-05
 5.89314850e-05 5.50953553e-05 5.38651291e-05 4.95058368e-05
 4.63910239e-05 4.12885342e-05 3.83893472e-05 3.58508778e-05
 3.49719620e-05 3.40729171e-05 3.32645356e-05 2.98640373e-05
 2.76652868e-05 2.68273706e-05 2.51677553e-05 2.43862103e-05
 2.21071647e-05 2.19900985e-05 1.98916321e-05 1.86899088e-05
 1.74432121e-05 1.57862924e-05 1.52566843e-05 1.46904708e-05
 1.43740637e-05 1.31077173e-05 1.22668460e-05 1.13367076e-05
 1.08714589e-05 9.84147999e-06 9.43655624e-06 8.92058938e-06
 8.61105875e-06 8.22777421e-06 7.66821258e-06 7.50565466e-06
 7.16226968e-06 6.20180358e-06 6.12791407e-06 5.48314145e-06
 5.14759712e-06 4.73200161e-06 4.36417780e-06 4.21852984e-06
 4.01154460e-06 3.78196441e-06 3.52842226e-06 3.35575340e-06
 3.11884565e-06 2.86973003e-06 2.50785526e-06 2.38270140e-06
 2.32503684e-06 2.14359190e-06 2.05350805e-06 1.94994418e-06
 1.78465996e-06 1.74457920e-06 1.60986778e-06 1.40372075e-06
 1.37966033e-06 1.27500488e-06 1.21339399e-06 1.14163807e-06
 1.01079415e-06 9.44060339e-07 8.67393244e-07 8.10879717e-07
 7.73455404e-07 7.23085128e-07 6.88147448e-07 6.32639235e-07
 5.96502311e-07 5.83033433e-07 5.22671544e-07 4.81561085e-07
 4.39902937e-07 4.34270298e-07 4.07889152e-07 3.65538013e-07
 3.25875306e-07 3.14458246e-07 2.76400529e-07 2.47351778e-07
 2.32743051e-07 2.03442467e-07 1.92685008e-07 1.69458073e-07
 1.59006935e-07 1.53841344e-07 1.43172073e-07 1.17414275e-07
 9.68222407e-08 9.47928466e-08 8.16371446e-08 8.00132657e-08
 6.32438208e-08 5.50554186e-08 5.18361078e-08 4.66745739e-08
 4.00332471e-08 3.21227256e-08 2.73939385e-08 1.93493170e-08
 1.92189456e-08 1.56200060e-08 1.20016986e-08 8.03262981e-09
 5.88574280e-09]
```

Now we have to plot for up to 10 dimensions.

```
var_10 = var[:10]
print(var_10)
```

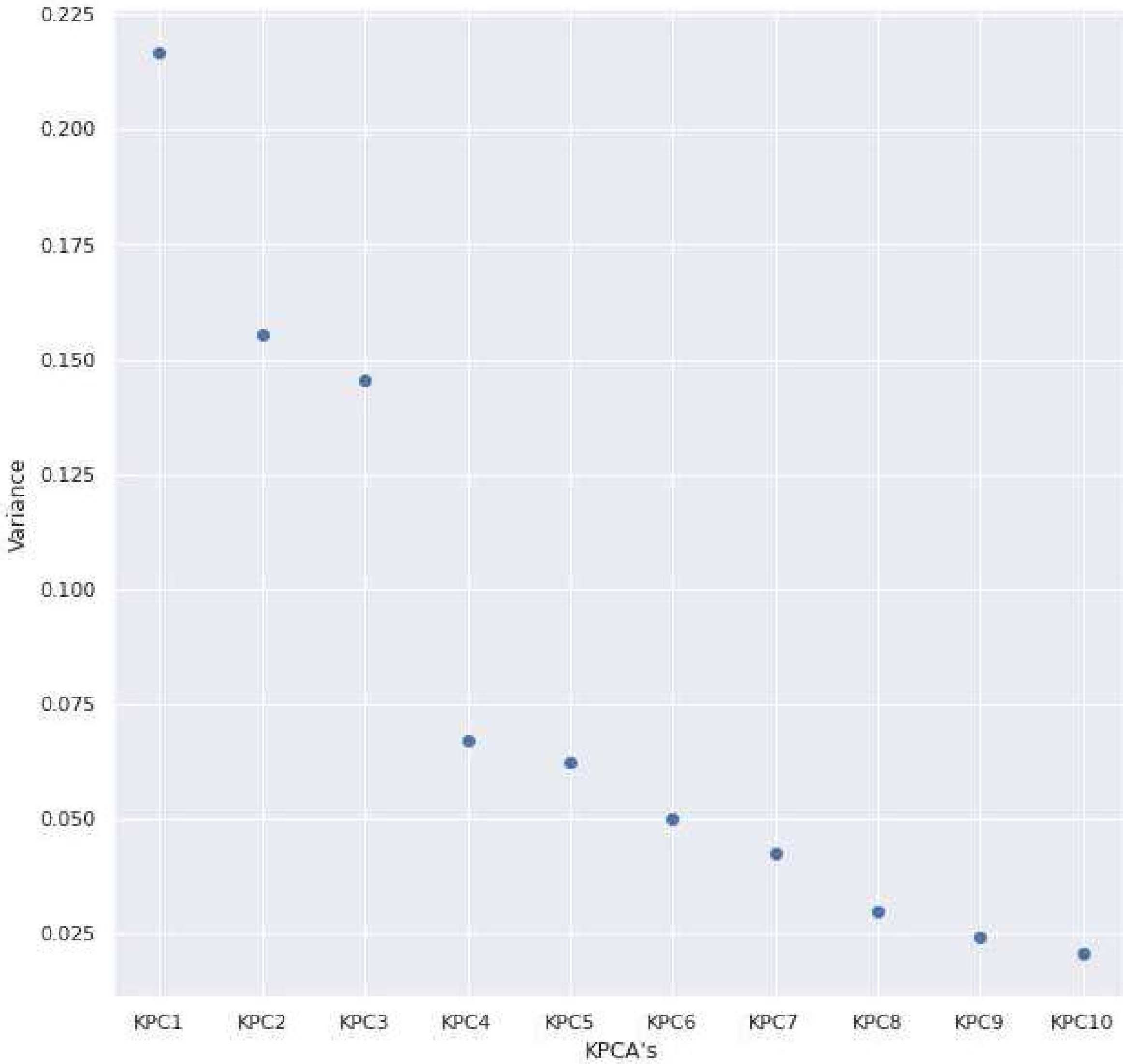
```
[0.21658246 0.15549357 0.14564044 0.066985   0.06242899 0.05014784
 0.04260795 0.02993025 0.02427455 0.0209806 ]
```

Observations

The explained variances are calculated above , they are coming out to be 21.65% , 15.54%, 14.56% , 6.66% , 6.24% , 5.01% , 4.26% , 2.99% , 2.427% , 2.098% respectively.

```
PCA_axis =['KPC1' , 'KPC2' , 'KPC3' , 'KPC4', 'KPC5' , 'KPC6' , 'KPC7' ,'KPC8','KPC9','KPC
plt.scatter(PCA_axis , var_10)
plt.xlabel("KPCA's")
plt.ylabel("Variance")
plt.show
```

```
<function matplotlib.pyplot.show>
```



Plot is shown in between variance explained versus KPCA dimensions for up to 10 dimensions.

213070024

Home Logout IITB Website

Roll no	213070024	**Name**	Abhishek Verma
Department	Electrical Engineering	**Program**	M.Tech (Control and Computing)

Performance Summary | Graduation Requirements | Personal Information | Forms/Requests | Payment

Academic Performance Summary

Year	Sem	SPI	CPI	Sem Credits Used for SPI	Completed Semester Credits	Cumulative Credits Used for CPI	Completed Cumulative Credits
2021	Spring	8.36	8.19	28.0	28.0	52.0	52.0
2021	Autumn	8.0	8.0	24.0	24.0	24.0	24.0

Semester-wise Details

*This registration is subject to approval(s) from faculty advisor/Course Instructor/Academic office.

Year/Semester: 2022-23/Autumn

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
SC 649	Embedded Control & Robotics	6.0	Department elective	Not allotted	C

Year/Semester: 2022-23/Project

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 797	I Stage Project	42.0	Core course	Not allotted	C

Year/Semester: 2021-22/Spring

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 613	Nonlinear Dynamical Systems	6.0	Core course	AB	C
EE 622	Optimal Control Systems	6.0	Core course	BB	C
EE 636	Matrix Computations	6.0	Core course	BB	C
EE 694	Seminar	4.0	Core course	AB	C
EE 769	Introduction to Machine Learning	6.0	Department elective	BB	C

GC 101	Gender in the workplace	0.0	Core course	PP	N

Year/Semester: 2021-22/Autumn

Course Code	Course Name	Credits	Tag	Grade	Credit/Audit
EE 601	Statistical Signal Analysis	6.0	Core course	BB	C
EE 615	Control and Computational Laboratory	6.0	Core course	AB	C
EE 635	Applied Linear Algebra	6.0	Core course	BC	C
EE 640	Multivariable Control Systems	6.0	Core course	BB	C
EE 899	Communication Skills	6.0	Core course	PP	N