SVD

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("test.csv")


#Perform SVD on the dataset
U,s,V=np.linalg.svd(df)


#construct the S amtrix
S=np.zeros((df.shape[0],df.shape[1]))
S[:df.shape[1], :df.shape[1]]=np.diag(s)


#Reconstruct  the original matrix
reconstructed_data=np.dot(U, np.dot(S,V))


#compute the rank-k approximation of the data
k=10
approx_data = np.dot(U[:, :k], np.dot(S[:k, :k], V[:k, :]))


# Print the results
print("Original data:\n", df)
print("\nReconstructed data:\n", reconstructed_data)
print("\nRank-k approximation:\n", approx_data)
```

```
Original data:
      profile pic  nums/length username  fullname words  nums/length fullname  \
0               1                  0.33               1                  0.33
1               1                  0.00               5                  0.00
2               1                  0.00               2                  0.00
3               1                  0.00               1                  0.00
4               1                  0.50               1                  0.00
..            ...                   ...             ...                   ...
115             1                  0.29               1                  0.00
116             1                  0.40               1                  0.00
117             1                  0.00               2                  0.00
118             0                  0.17               1                  0.00
119             1                  0.44               1                  0.00

     name==username  description length  external URL  private  #posts  \
0                 1                  30             0        1      35
1                 0                  64             0        1       3
2                 0                  82             0        1     319
3                 0                 143             0        1     273
4                 0                  76             0        1       6
..              ...                 ...           ...      ...     ...
115               0                   0             0        0      13
116               0                   0             0        0       4
117               0                   0             0        0       3
118               0                   0             0        0       1
119               0                   0             0        0       3

     #followers  #follows  fake
0           488       604     0
1            35         6     0
2           328       668     0
3         14890      7369     0
4           225       356     0
..          ...       ...   ...
115         114       811     1
116         150       164     1
117         833      3572     1
118         219      1695     1
119          39        68     1

[120 rows x 12 columns]

Reconstructed data:
 [[ 1.00000000e+00  3.30000000e-01  1.00000000e+00 ...  4.88000000e+02
   6.04000000e+02  1.46123062e-14]
 [ 1.00000000e+00 -1.00708232e-11  5.00000000e+00 ...  3.50000000e+01
   6.00000000e+00  4.51994827e-14]
 [ 1.00000000e+00  3.25093758e-11  2.00000000e+00 ...  3.28000000e+02
   6.68000000e+02  1.38044365e-14]
 ...
 [ 1.00000000e+00 -3.15990164e-11  2.00000000e+00 ...  8.33000000e+02
   3.57200000e+03  1.00000000e+00]
 [-1.18574833e-14  1.70000000e-01  1.00000000e+00 ...  2.19000000e+02
   1.69500000e+03  1.00000000e+00]
```

```
 [ 1.00000000e+00  4.40000000e-01  1.00000000e+00 ...  3.90000000e+01
   6.80000000e+01  1.00000000e+00]]

Rank-k approximation:
```

PCA

```
def PCA(X , num_components):

    #Step-1
    X_meaned = X - np.mean(X , axis = 0)

    #Step-2
    cov_mat = np.cov(X_meaned , rowvar = False)

    #Step-3
    eigen_values , eigen_vectors = np.linalg.eigh(cov_mat)

    #Step-4
    sorted_index = np.argsort(eigen_values)[::-1]
    sorted_eigenvalue = eigen_values[sorted_index]
    sorted_eigenvectors = eigen_vectors[:,sorted_index]

    #Step-5
    eigenvector_subset = sorted_eigenvectors[:,0:num_components]

    #Step-6
    X_reduced = np.dot(eigenvector_subset.transpose() , X_meaned.transpose() ).transpose()

    return X_reduced


#prepare the data
x = df.iloc[:, df.columns != 3]

#prepare the target
target = df.iloc[:,3]

#Applying it to PCA function
mat_reduced = PCA(x , 2)

#Creating a Pandas DataFrame of reduced Dataset
principal_df = pd.DataFrame(mat_reduced , columns = ['PC1','PC2'])

#Concat it with target variable to create a complete Dataset
principal_df = pd.concat([principal_df , pd.DataFrame(target)] , axis = 1)
```

```
df
```

	profile pic	nums/length username	fullname words	nums/length fullname	name==username	description length	external URL	private	#post
0	1	0.33	1	0.33	1	30	0	1	3
1	1	0.00	5	0.00	0	64	0	1	
2	1	0.00	2	0.00	0	82	0	1	31
3	1	0.00	1	0.00	0	143	0	1	27
4	1	0.50	1	0.00	0	76	0	1	
...	...	...	...	...	...	...	...	...	
115	1	0.29	1	0.00	0	0	0	0	1
116	1	0.40	1	0.00	0	0	0	0	
117	1	0.00	2	0.00	0	0	0	0	
118	0	0.17	1	0.00	0	0	0	0	
119	1	0.44	1	0.00	0	0	0	0	

120 rows × 12 columns

```
principal_df
```

	PC1	PC2	nums/length fullname
0	-49106.893857	-122.676333	0.33
1	-49560.550015	-720.260368	0.00
2	-49266.685661	-56.297192	0.00
3	-34697.519279	6628.705797	0.00
4	-49370.172378	-370.407992	0.00
...	...	...	...
115	-49480.683029	84.446306	0.00
116	-49445.382256	-562.634478	0.00
117	-48758.723191	2844.509179	0.00
118	-49374.739538	968.219105	0.00
119	-49556.485763	-658.518744	0.00

120 rows × 3 columns

✓ 0s completed at 10:58 PM