ADVANCED ALGORITHMS

PROJECT 2 - *IMAGE COMPRESSION USING SVD AND DIMENSIONALITY REDUCTION USING PCA*

Guided by,

Dr: Zhong-Hui Duan

Team Members

- Bini Elsa Paul
- Geethika Lakshi Yarra
- Olaa Kasem

Contents

INTRODUCTION

- We spent one week to study the project requirement.
- All of us came with ideas to approach the problem / project
- We have divided the project into 3 parts
  - Saving ASCII pgm to Binary and Reversal – Olaa
  - Image compression and floating point to binary – Bini
  - PCA –Geethika
- Decided to work on JAVA
- We worked independently
- Communicated doubts / problems faced / shared useful web sites
- Integrated our parts by March 31

Part 1: Ascii file to binary file

- The objective is to read an ascii pgm file and convert it to binary pgm file with reducing its size.
- Save all information in a binary array and write this array to output file

1. Reading the whole file
2. Ignore the magic number and the comment line because we are not going to save it in the binary file
3. Store the number of rows and columns of the image pixels array in 2 byte each

```
barray[0] = (byte)(Rows & 0xff);
barray[1] = (byte)((Rows >> 8) & 0xff);
```

4. Store the grey scale levels number and each element of pixels array in 1 byte

```
bvalue = (byte)(value & 0xff);
```

- The size of new file will be (Rows*Columns)+5 byte

Binary file to Ascii file

- The objective is reverse what we have done.

1. Reading the binary array from the binary file
2. Read first 2 bytes from the array and store it as width of the image and 3rd and 4th bytes as height of the image

```
short width = (short) ((data[1] << 8) + (data[0]&0xFF));
 short height= (short) ((data[3] << 8) + (data[2]&0xFF));
```

3. Read the 5th byte from the array as the grey scale levels and the rest of the array will be the pixels

```
ivalue = bvalue & 0xff;
```

4. Write all these values to the new file

Application

- Reduce the size of the image file by reducing the space needed to store each element in the file
- Reduces the bandwidth to transmit the data/image

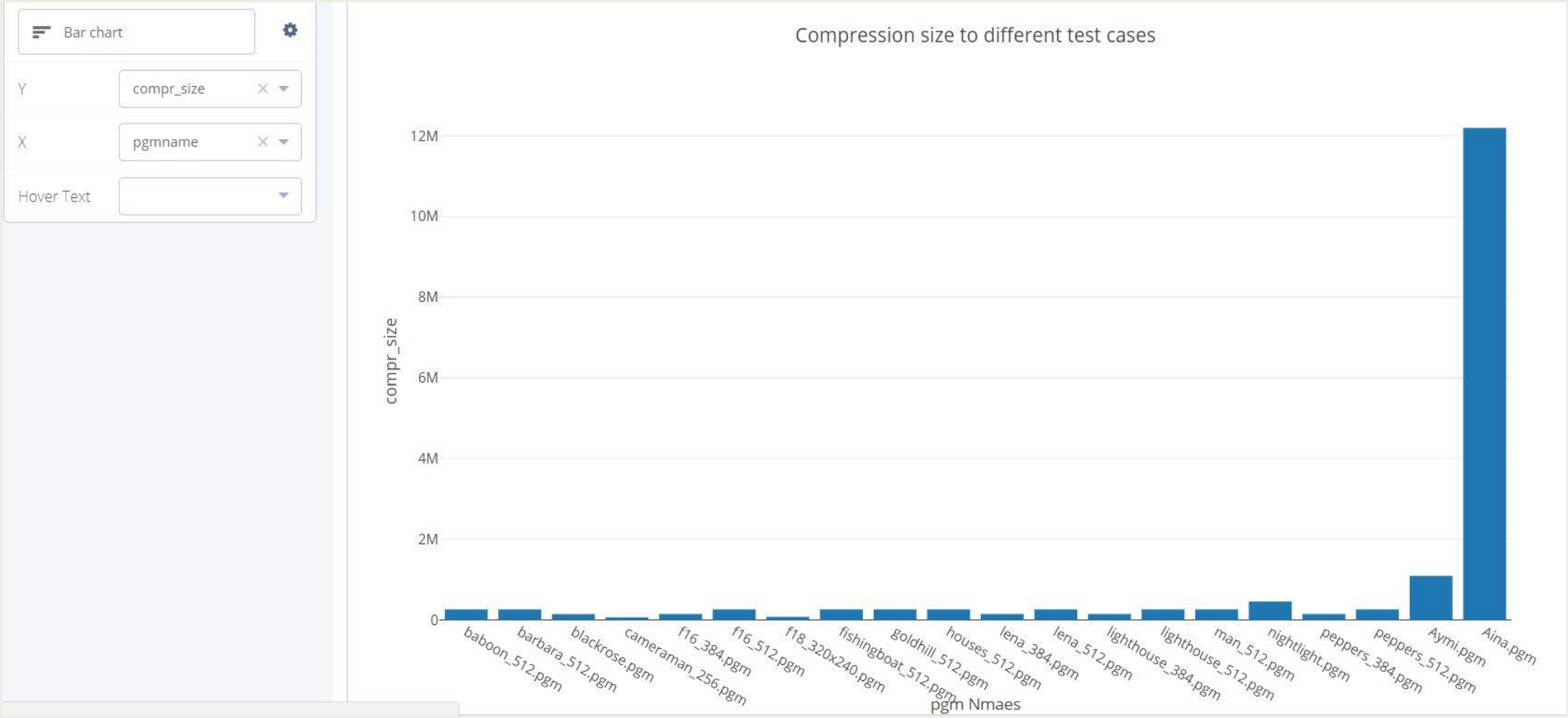
Test cases and results

- We have tested this part over 20 different image files with different sizes
  - 4 X 5 to 3024 X 4032 pixels
- All results was as expected

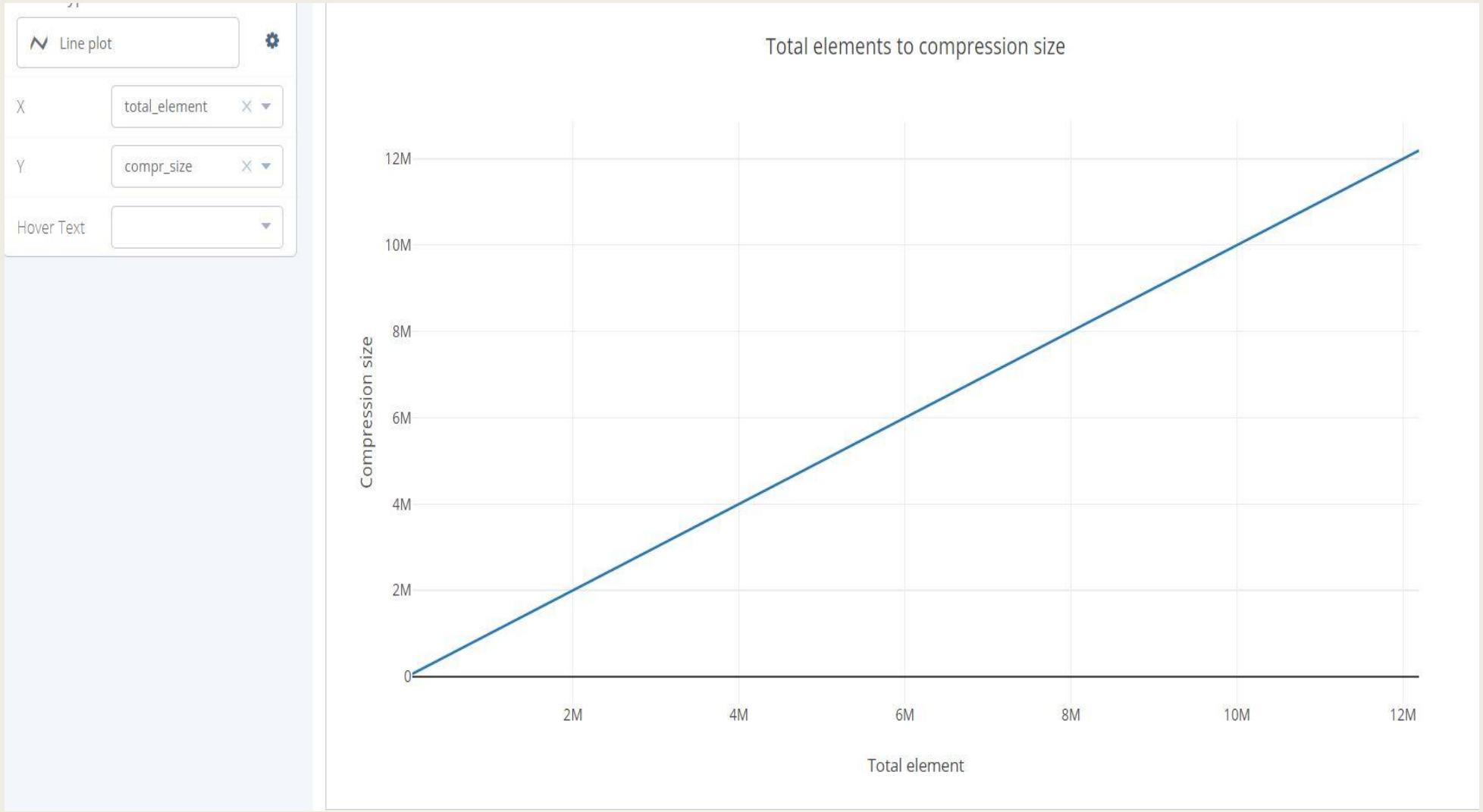
Results for different test cases

	A	B	C	D	E	F	G	H	I	J
1	id	pgmname	org_size	compr_siz	recover_si	rateof_cor	rows	columns	total_element	
2	1	baboon_5	992655	262149	977783	0.735911	512	512	262144	
3	2	barbara_5	963732	262149	948860	0.727986	512	512	262144	
4	3	blackrose.	514429	144405	506425	0.719291	380	380	144400	
5	4	camerama	245442	65541	242090	0.732967	256	256	65536	
6	5	f16_384.p	584605	147461	576517	0.74776	384	384	147456	
7	6	f16_512.p	1038222	262149	1023350	0.747502	512	512	262144	
8	7	f18_320x2	288578	76805	284634	0.73385	320	240	76800	
9	8	fishingboa	1008797	262149	993925	0.740137	512	512	262144	
10	9	goldhill_5	950610	262149	935738	0.724231	512	512	262144	
11	10	houses_51	961715	262149	946843	0.727415	512	512	262144	
12	11	lena_384.p	552560	147461	544472	0.733131	384	384	147456	
13	12	lena_512.p	983406	262149	968534	0.733427	512	512	262144	
14	13	lighthouse	530391	147461	522303	0.721977	384	384	147456	
15	14	lighthouse	944168	262149	929296	0.722349	512	512	262144	
16	15	man_512.p	892432	262149	877560	0.706253	512	512	262144	
17	16	nightlight.	1406011	457605	1379131	0.674537	572	800	457600	
18	17	peppers_3	537661	147461	529573	0.725736	384	384	147456	
19	18	peppers_5	957234	262149	942362	0.726139	512	512	262144	
20	19	Aymi.pgm	4005578	1091845	3939420	0.727419	853	1280	1091840	
21	20	Aina.pgm	43722475	12192773	42970483	0.721133	3024	4032	12192768	

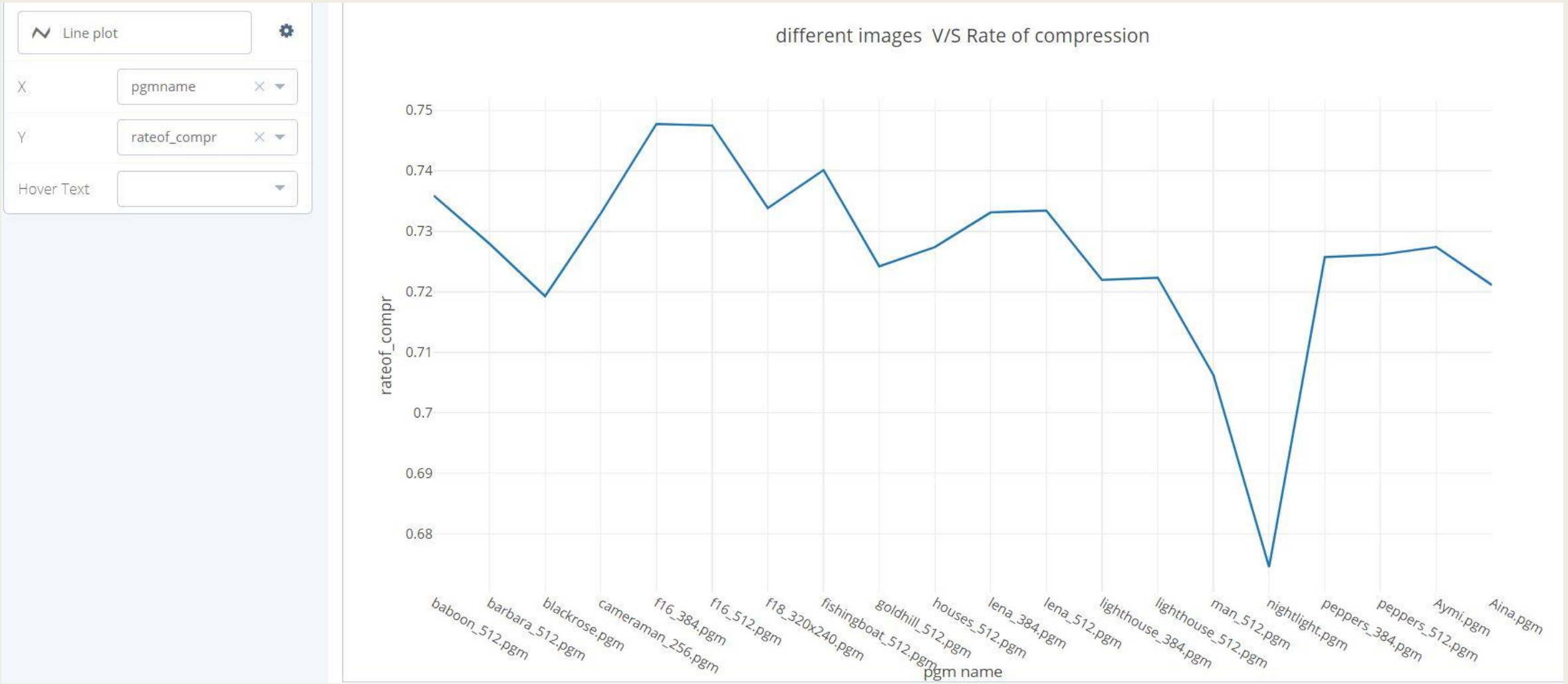
Results


Bar chart
Y
compr_size
X
pgmname
Hover Text
Compression size to different test cases
12M
10M
8M
6M
4M
2M
0
compr_size
baboon_512.pgm
barbara_512.pgm
blackrose.pgm
cameraman_256.pgm
f16_384.pgm
f16_512.pgm
f18_320x240.pgm
fishingboat_512.pgm
goldhill_512.pgm
houses_512.pgm
lena_384.pgm
lena_512.pgm
lighthouse_384.pgm
lighthouse_512.pgm
man_512.pgm
nightlight.pgm
peppers_384.pgm
peppers_512.pgm
Aymi.pgm
Aina.pgm
pgm Nmaes

Results


Line plot
X
total_element
Y
compr_size
Hover Text
Total elements to compression size
12M
10M
8M
6M
4M
2M
0
Compression size
2M
4M
6M
8M
10M
12M
Total element

Results


Line plot
X
pgmname
Y
rateof_compr
Hover Text
different images V/S Rate of compression
rateof_compr
0.75
0.74
0.73
0.72
0.71
0.7
0.69
0.68
baboon_512.pgm
barbara_512.pgm
blackrose.pgm
cameraman_256.pgm
f16_384.pgm
f16_512.pgm
f18_320x240.pgm
fishingboat_512.pgm
goldhill_512.pgm
houses_512.pgm
lena_384.pgm
lena_512.pgm
lighthouse_384.pgm
lighthouse_512.pgm
man_512.pgm
nightlight.pgm
peppers_384.pgm
peppers_512.pgm
Aymi.pgm
Aina.pgm
pgm name

Part 2: SVD Image compression

Problem Statement:

The objective of this part is to understand how SVD works in the application of image compression. Input to this part will be the header information, SVD of a pgm image format and a K value. The problem is to save only the necessary information in binary format and to recover the compressed image. The logic of this part is

Image = U*Sigma*VT

I have divided this part to different sub parts

- Getting SVD of a pgm image file and finding pgm file viewer
- Converting double to 16 bit (Encoding)
- Writing and reading binary file
- Decode compressed float values from binary to double
- Get the Compressed image
- Find the Error image
- Testing and analyzing the results

Algorithm

1. Read the input image
2. Convert it into gray scale image. (IrfanView)
3. Decompose the image using singular value decomposition. (get SVD – Apache Commons Math)
4. Discard the singular values not required for compression. (Use only K singular values)
5. Convert the doubles to 2 byte binary
6. Write the compressed file
7. Read binary file
8. Decode binary to double
9. Reconstruct the image and get the error image.
10. Compute compression ratio and MSE.

Getting SVD of a pgm image file

Got two options to get the SVD of a given matrix

1. Using JAMA package

   Problem faced: Array Index out of bound Exception if rows < columns

   Old package

2. Apache Commons Math

   Problem faced: Array Index out of bound Exception if rows < columns

   Solution : Used Transpose of the input image

   Problem faced: U and V matrix size different if rows > columns and thus Missing some part of the image

   Reason : For test case 320 * 240, the generated SVD is

   Sigma – 240

   U – 240 * 240

   V – 320 * 240

   U*Sigma*VT -> 240 * 320 - Missing some rows

   Solution : Interchanged U and V and took Transpose of the recovered image

Pgm File Viewer : - Irfan View 64

- Opens pgm image file
- Converts JPG image to pgm ASCII coded image

Converting double to 16 bit (Encoding)

First Approach

Find out the lease and most possible values for SVD(max : 4 digits)

Sign Bit	Exponential – 3 bits			Mantissa – 12 bits											
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

- Saving maximum bits for mantissa
- Bring all the digits before the decimal point
  - 12.36 - > 1236 (divide by 10 - 2 times)
  - 0.0235 -> 235 (divide by 10 - 4 times)
  - 165 -> 165 (divide by 10 - 0 times)

- Can save the sign bit for the exponential at 2^14 (can store 2^14 – 2^13 more numbers)

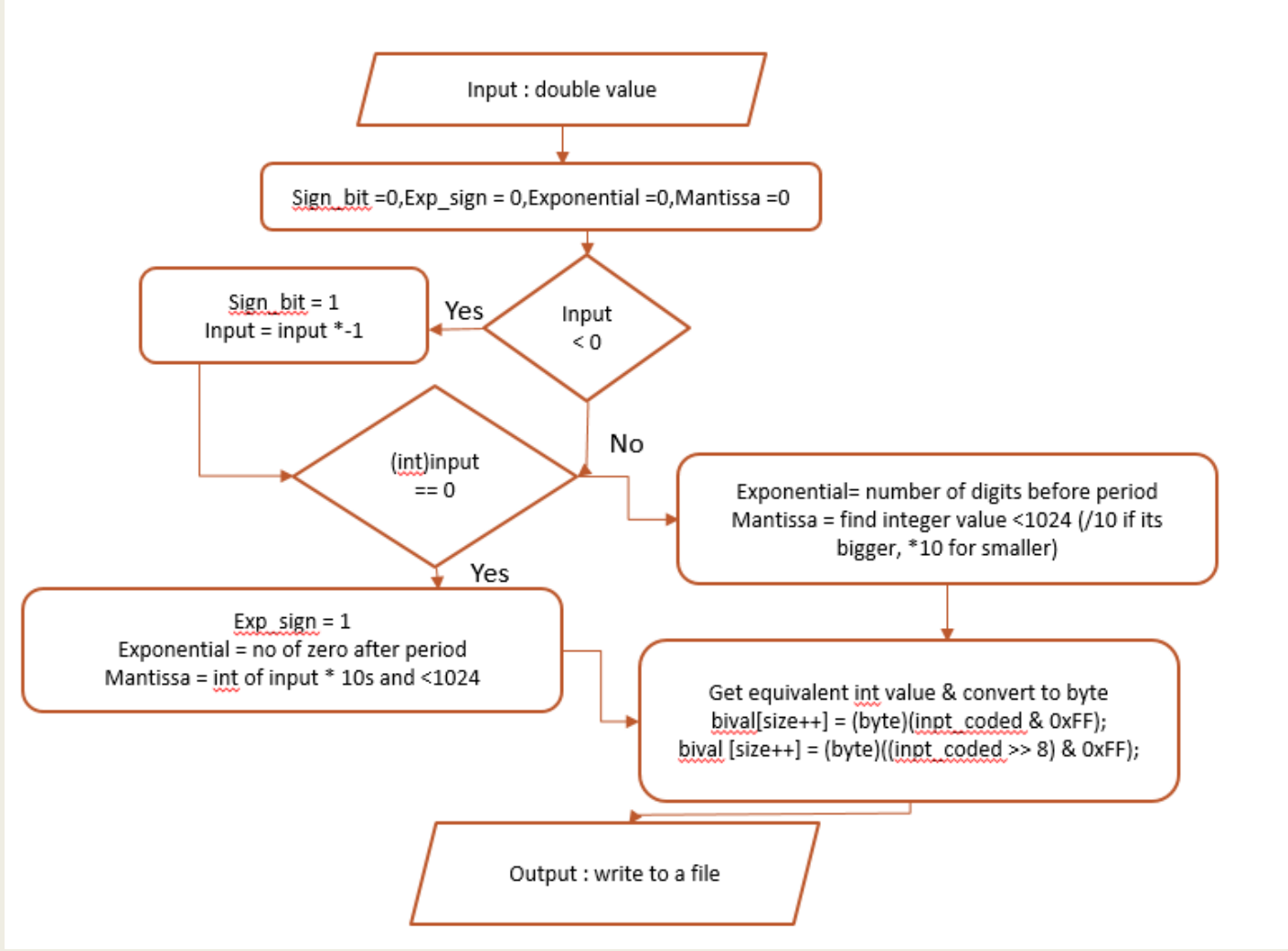
Problem faced: SVD with 6 digit

324568 :- 3245 (Multiply by 10 - 2 times)

Ended up with using sign bit for exponent

Converting double to 16 bit (Encoding)

Sign	Exp Sign	Exponential			Mantissa										
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0



```
switch(exp_value){
case 0:
element_coded = (int)Math.pow(2, 15)*sign + mantissa;
break;
case 1:
element_coded = (int)Math.pow(2, 15)*sign + (int)Math.pow(2, 14)*exp_value_sign +(int)Math.pow(2, 11) + mantissa;
break;
case 2:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign+(int)Math.pow(2, 12) + mantissa;
break;
case 3:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign +(int)Math.pow(2, 12)
+(int)Math.pow(2, 11)+ mantissa;
break;
case 4:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign+(int)Math.pow(2, 13) + mantissa;
break;
case 5:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign+(int)Math.pow(2,
13)+(int)Math.pow(2, 11) + mantissa;
break;
case 6:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign+(int)Math.pow(2,
13)+(int)Math.pow(2, 12) + mantissa;
break;
case 7:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign +(int)Math.pow(2,
13)+(int)Math.pow(2, 12)+(int)Math.pow(2, 11) + mantissa;
break;
default:
element_coded = (int)Math.pow(2, 15)*sign  + (int)Math.pow(2, 14)*exp_value_sign +(int)Math.pow(2,
13)+(int)Math.pow(2, 12)+(int)Math.pow(2, 11) + mantissa;
}
```

Writing and reading binary file

Convert to Binary

- bival[size++] = (byte)(inpt_coded & 0xFF);
- bival [size++] = (byte)((inpt_coded >> 8) & 0xFF);

Writing

- Path path = Paths.*get(file_path+svd_name+".pgm.SVD");*
- Files.*write(path, bival);*

The size of the compressed file will be 2*k*(1+rows+colums);

Reading

- Path path = Paths.*get(file_path+binary_svd);*
- byte[] data = Files.*readAllBytes(path);*

Convert to Integer

- high = data[1] >= 0 ? data[1] : 256 + data[1];
- low = data[0] >= 0 ? data[0] : 256 + data[0];
- int result = low | (high << 8);

Decoding Value

- Convert to String
- Get its length, prefix zeros till length is 16
- Sign = substring of 0,1
- Exp_sign = substring of 1,2
- Exp = substring 2,5
- Mantissa = substring 5, end
- Convert all to int from binary
- Get mantissas all digits after period (/10)
- Exp_sign is 1 then multiplier is Math.*pow(10, power*-1) otherwise* Math.*pow(10, power)*
- *Sign is 1 then multiply by -1*

Get the Compressed image

- Store the decoded values in sigma, U and V.
- Sigma - k * k size
- U - rows * k
- V - columns * k
- Get the transpose of V
- Multiply Sigma, U, VT
- Take transpose if rows != columns
- Write to a file (convert double to int)
- Used PrintWriter class

Find the Error image

- Original image – compressed image
- Original SVD– compressed SVD
- Mean Square Error – (Sum of squares of each element in the error matrix) / (rows*columns)

Testing and analyzing the results

1. Incorporated all the functionality to one function.
2. Created a table for compressed image details in MySQL with attributes, pgm name, K_value, compression ratio, MSE…..
3. Stored all the 20 test cases in a string array.
4. Looped through these test cases for k_values 25 to k_value < row for 20 test cases
5. Calculated compression ratio, and MSE
6. Inserted these values to the DB.
7. Exported values from DB to csv format
8. Plotted the graphs using online tool.

- **Compression Ratio:**

  It is defined as the ratio of original size of the image to compressed size of the image. It gives measure of the degree to which an image is compressed.

  Compression ratio = (Size of original SVD file/Size of compressed SVD file)

- **Mean Square Error (MSE):**

  It gives the measure of degradation of compressed image quality as compared to the original image. It is defined as square of the difference between original image pixel values and the corresponding pixel value of the compressed image, averaged over the entire image.

  $$MSE = (\sum\sum(g(x,y) - g''(x,y))2) / (m*n);$$

  where m*n gives total number of pixels present in original image; g(x,y) and g''(x,y) are the pixel value of original and compressed image respectively.

Testing and analyzing the results

Total Test cases :20 +
Biggest Test Case : 3024 * 4032
Types of test cases : square matrix
rows < columns
columns < rows
Types of images: pgm, jpg (converted to pgm)
Result : Passed all test cases

Analysis : Compression ratio – Result as expected
Mean Square Error – Mixed Results
-MSE negative for one test case egs: blackrose
K < (rows*columns) / (rows + columns+1)

Analyzing the failure

The failure of MSE for some test cases can be explained with the precision error. This test case gives expected output for MSE without compression (storing double in 2 byte). When checked for the precision difference, it shows more than 40 for the highest sigma value. As the number of sigma used increases the precision error increases resulting in increasing MSE.

Implementation Difficulties

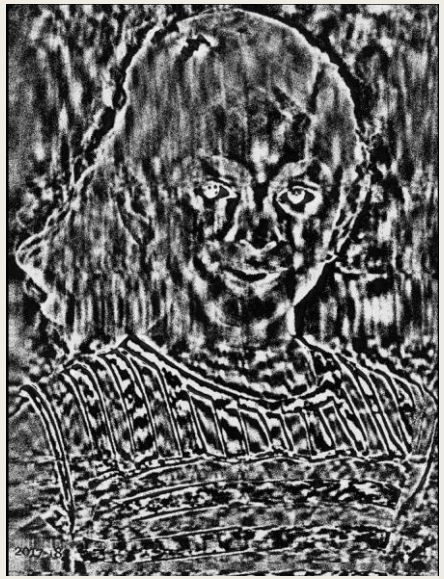
- Problems with getting SVD when rows != columns
- Failure of first approach to convert double to binary
- Got infinite loop for SVD value -0.0
- Missing part of the image if columns < rows (320 * 240) – Fixed the problem by interchanging rows and columns and taking transpose of the final image
- SVD is very slow for bigger test cases

Original image

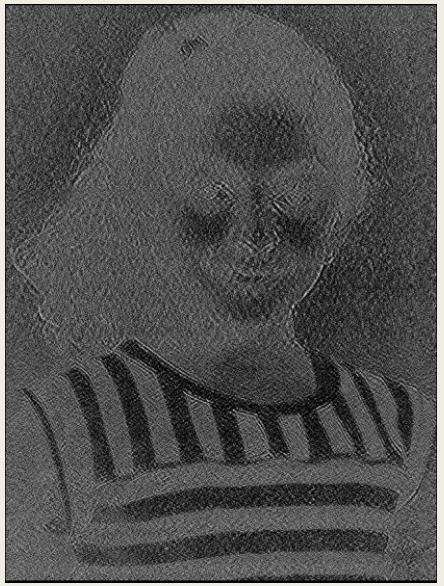
Corrected image

Screen Shots (SVD error image and no abs)

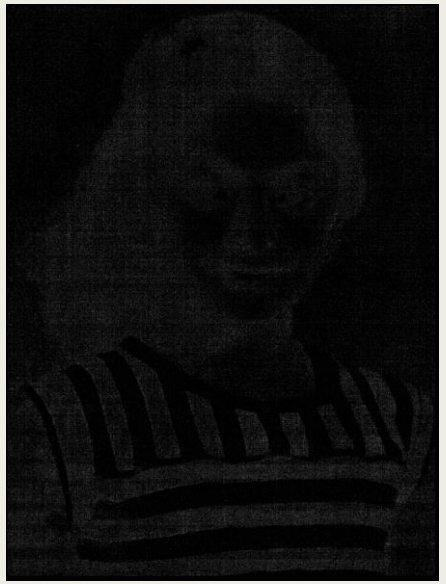
Greyscale : 3024 * 4032



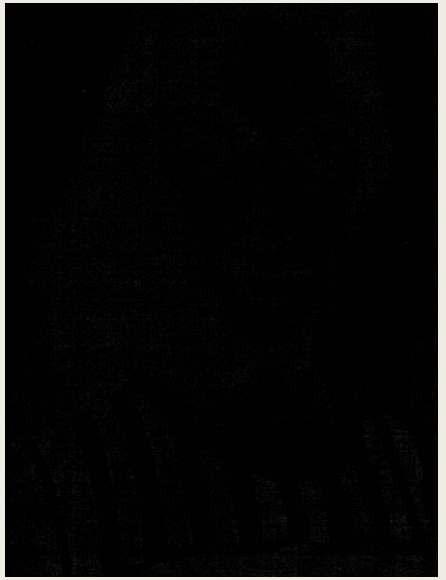
K = 25



K = 125

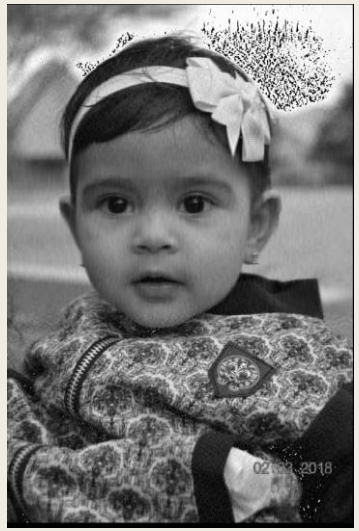
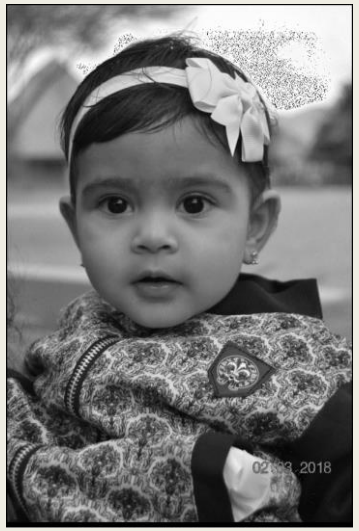
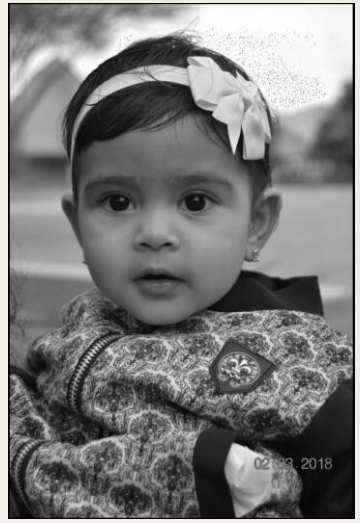
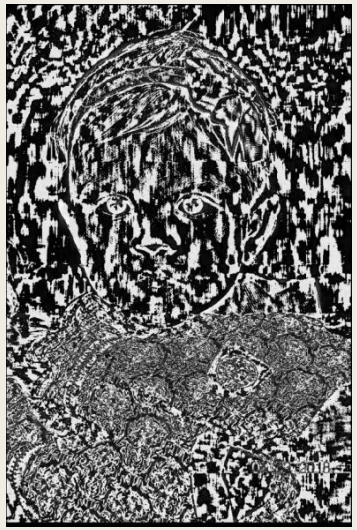
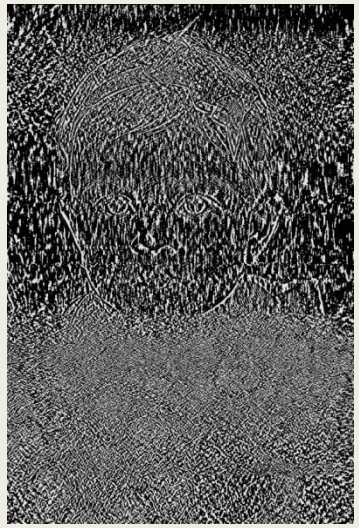
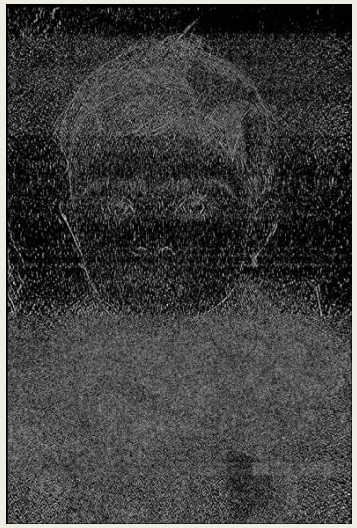
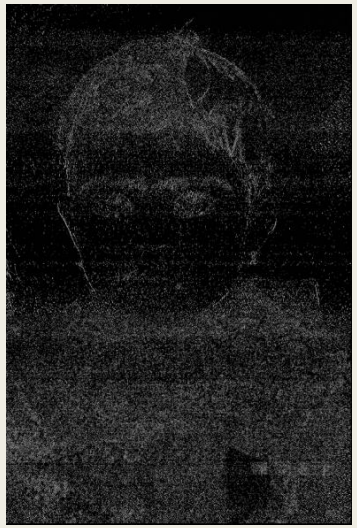
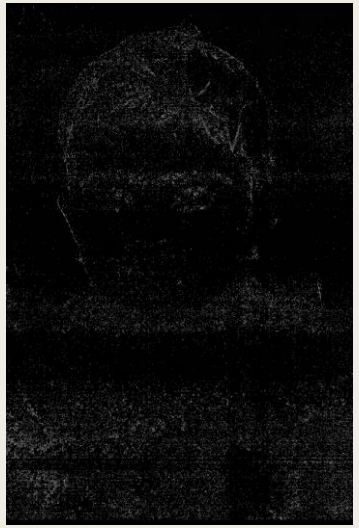
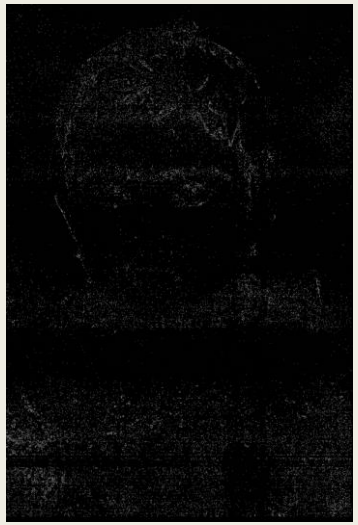


K = 1125



K = 1425

Screen Shots (SVD error image and no abs)

	K = 25	K = 100	K = 250	K = 400	K = 500	K = 525
						
						

Greyscale : 853 * 1280

Results

Aymi

k_value	MSE	compr_svd_ratio
25	347.9494	361.3801593
50	195.8221	180.6900797
75	125.7802	120.4600531
100	85.49798	90.34503983
125	60.17514	72.27603187
150	43.50317	60.23002655
175	32.04385	51.62573705
200	23.89407	45.17251992
225	17.99567	40.15335104
250	13.69474	36.13801593
275	10.47556	32.85274176
300	8.049278	30.11501328
325	6.239775	27.79847379
350	4.865606	25.81286852
375	3.819006	24.09201062
400	3.021641	22.58625996
425	2.418124	21.25765643
450	1.963117	20.07667552
475	1.614269	19.02000839
500	1.350806	18.06900797
525	1.149173	17.20857902
550	0.997106	16.42637088
575	0.881218	15.71218084
600	0.795977	15.05750664
625	0.731494	14.45520637

nightlight

k_value	compr_pgm_size	compr_svd_size	MSE	org_pgm_size	org_svd_size	rows	colums	compr_svd_ratio
25	1429295	68650	128.561844	1406011	16561014	572	800	241.2383685
75	1422160	205950	45.17282112	1406011	16561014	572	800	80.41278951
125	1418567	343250	21.65335009	1406011	16561014	572	800	48.24767371
175	1416687	480550	10.84995126	1406011	16561014	572	800	34.46262408
225	1415662	617850	5.346163368	1406011	16561014	572	800	26.80426317
275	1414898	755150	2.499307739	1406011	16561014	572	800	21.93076078
325	1414649	892450	1.08503312	1406011	16561014	572	800	18.55679758
375	1414475	1029750	0.424451488	1406011	16561014	572	800	16.0825579
425	1414385	1167050	0.157261024	1406011	16561014	572	800	14.19049227
475	1414085	1304350	0.069282211	1406011	16561014	572	800	12.69675624
525	1413649	1441650	0.049234486	1406011	16561014	572	800	11.48754136

Results

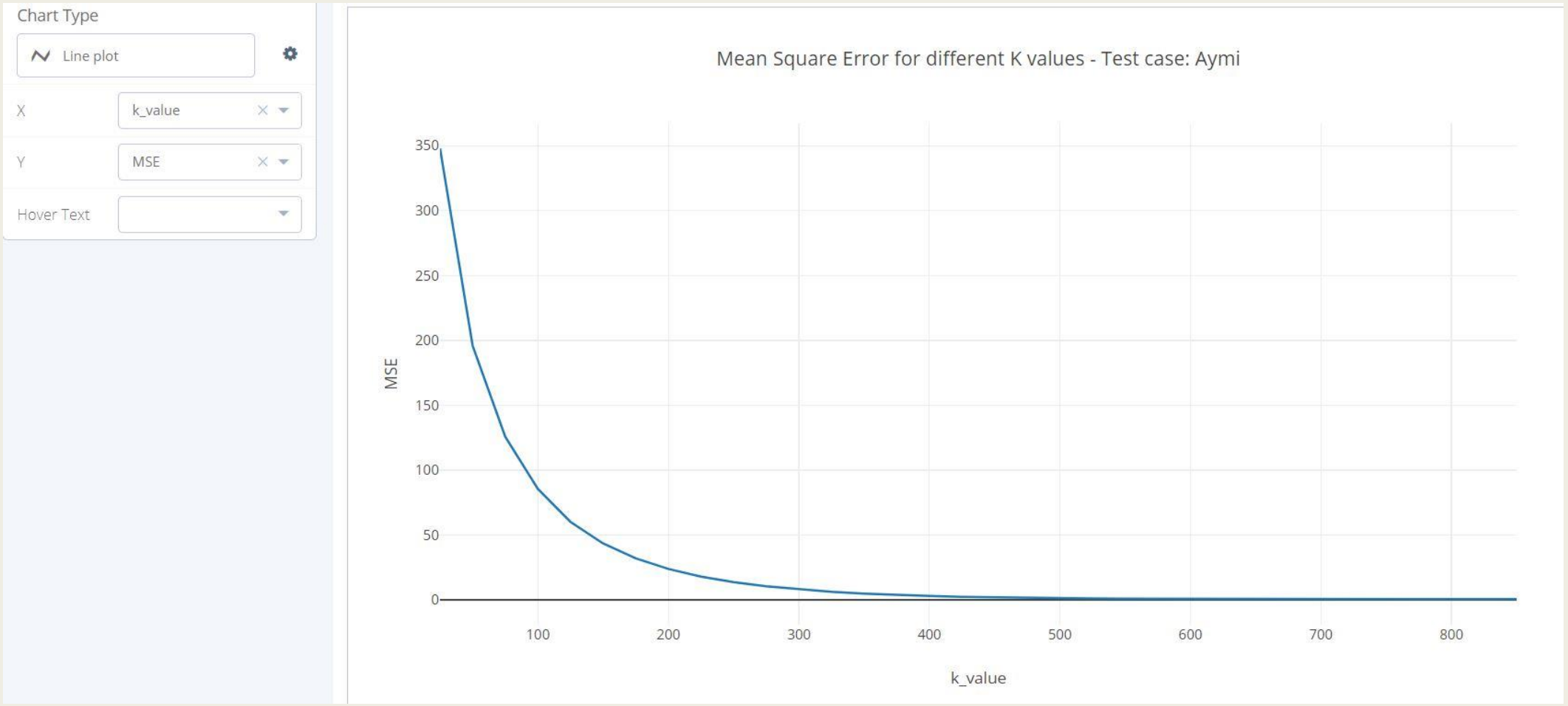
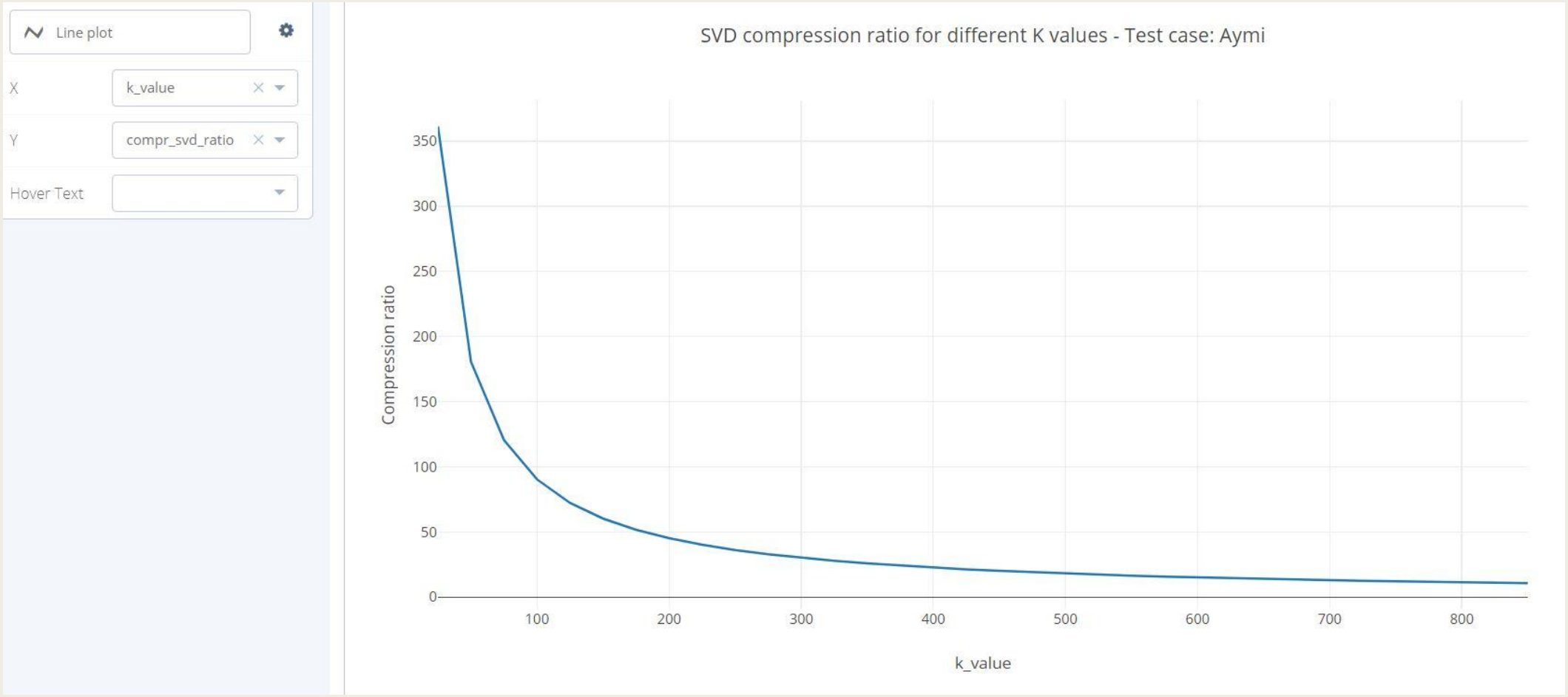
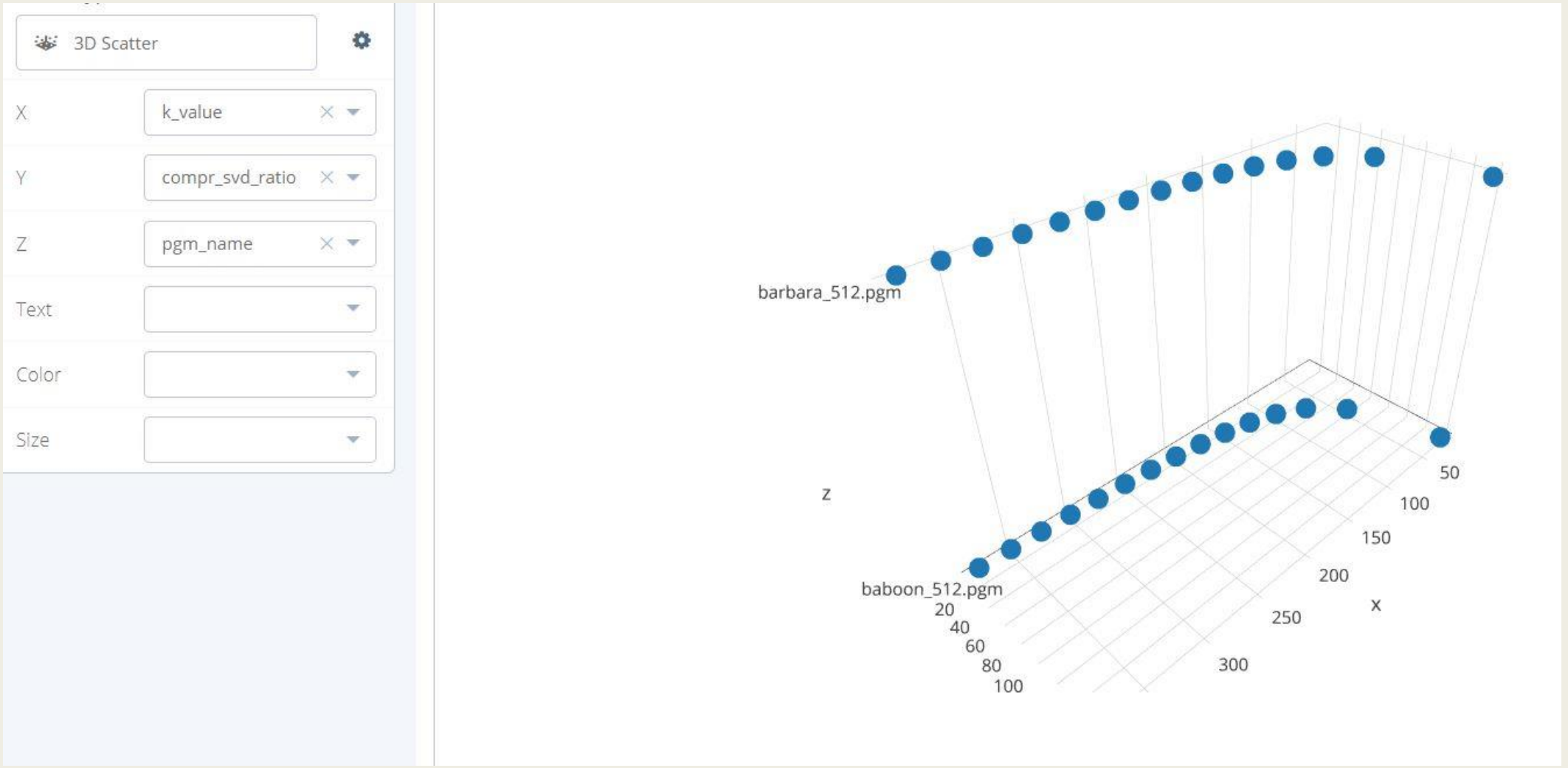


Chart Type
Line plot
X
k_value
Y
MSE
Hover Text
Mean Square Error for different K values - Test case: Aymi
MSE
350
300
250
200
150
100
50
0
100
200
300
400
500
600
700
800
k_value

Results

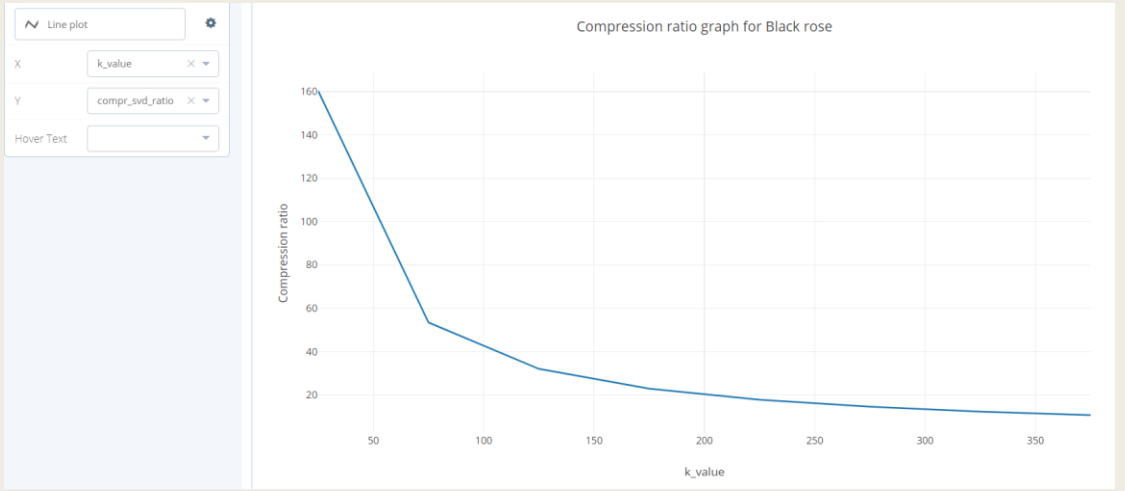

Line plot
X
k_value
Y
compr_svd_ratio
Hover Text
SVD compression ratio for different K values - Test case: Aymi
Compression ratio
350
300
250
200
150
100
50
0
100
200
300
400
500
600
700
800
k_value

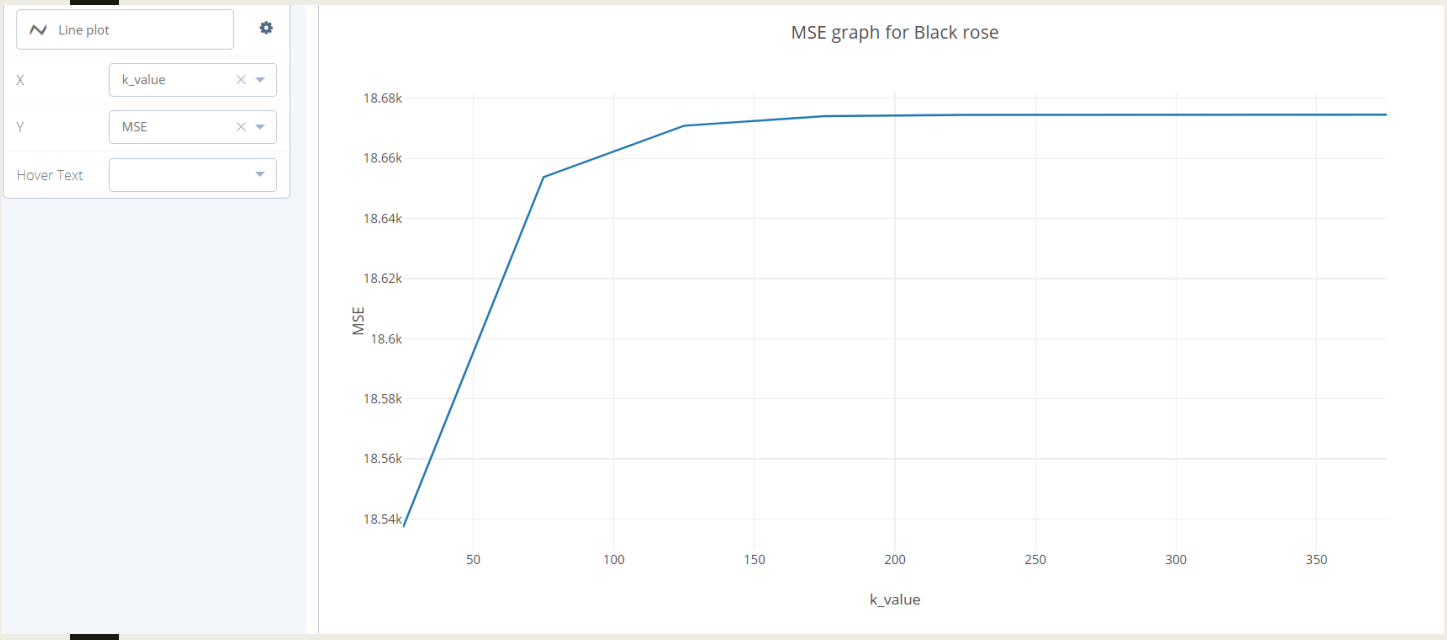
Results


3D Scatter
X
k_value
Y
compr_svd_ratio
Z
pgm_name
Text
Color
Size
barbara_512.pgm
baboon_512.pgm
z
20
40
60
80
100
300
250
200
150
100
50
x

Results - Exception

k	before	after	Difference	Square
1	67440.92	67400	40.91969	1674.421
5	67440.92	67400	40.91969	1674.421
5	21980.4	21900	80.40049	6464.238
5	10604.97	10600	4.973426	24.73497
5	5405.096	5400	5.096041	25.96963





k_value	MSE	compr_svd_ratio
25	18537.21	160.2846518
75	18653.77	53.42821726
125	18670.88	32.05693035
175	18674.03	22.8978074
225	18674.5	17.80940575
275	18674.57	14.57133198
325	18674.58	12.3295886

Application

SVD

- Image Compression
- Principal Component Analysis
- Data Reduction
- Eigen Faces
- Multi-Dimensional Scaling

Conclusion

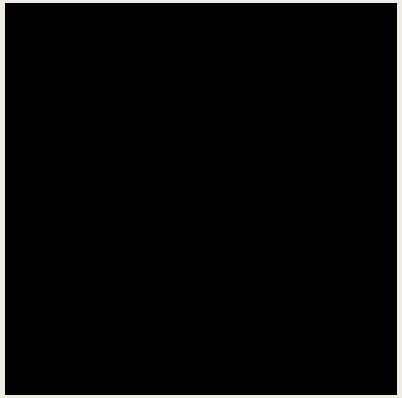
The task to use SVD for compressing images and storing double to 2 byte is achieved successfully. All the test cases passed for this task and few test cases gave unexpected result for MSE (egs: Blackrose)

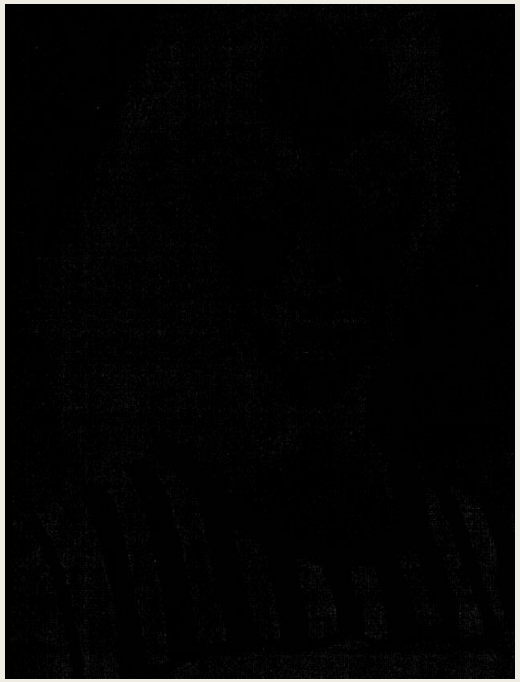
Findings:

- K < (rows*columns) / (rows + columns+1)
- Size of binary SVD = 2 * k (1+rows+columns)
- Saving double in 2 bytes for large images has less effect(precision errors) than small images. Small images has large error image even for large k_values.

256 * 256, K= 250

Without compression. K = 250



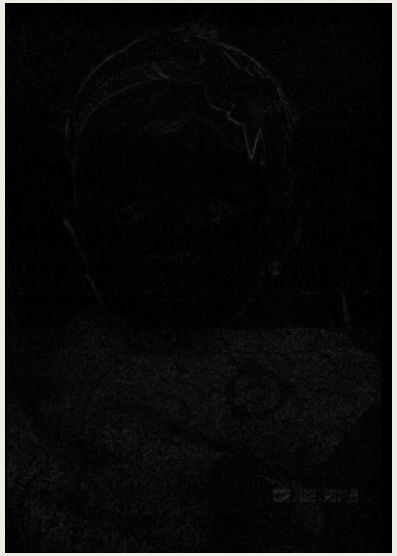


3024*4032 K=1425

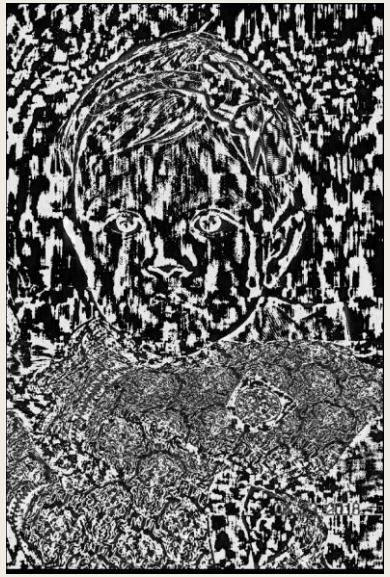
Conclusion(Error image with and without abs value)

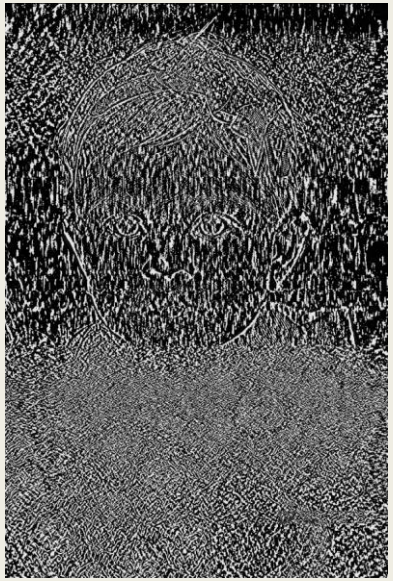
K = 25

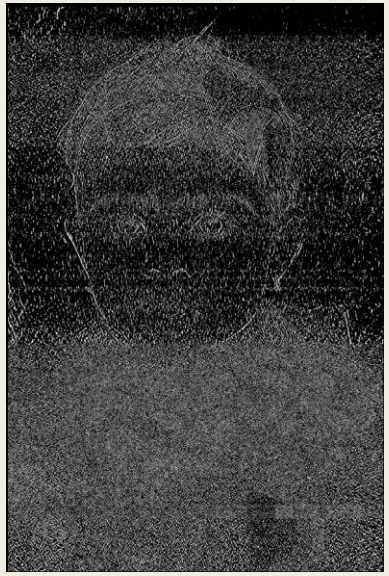
K = 100

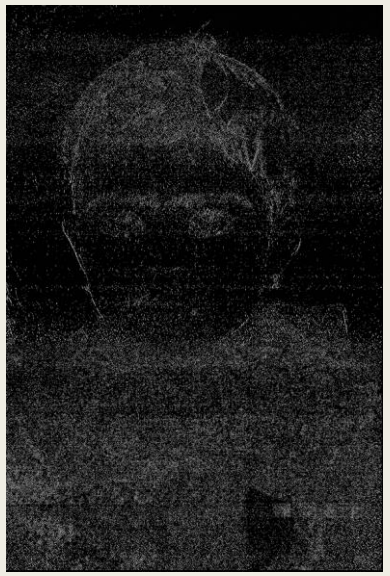


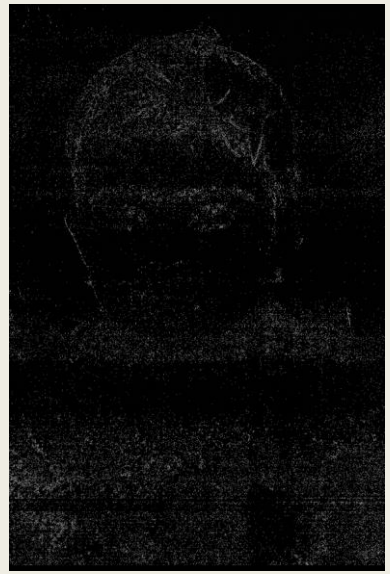
Top: Error with abs() value
Bottom: Error without abs()











Greyscale : 853 * 1280

K = 250

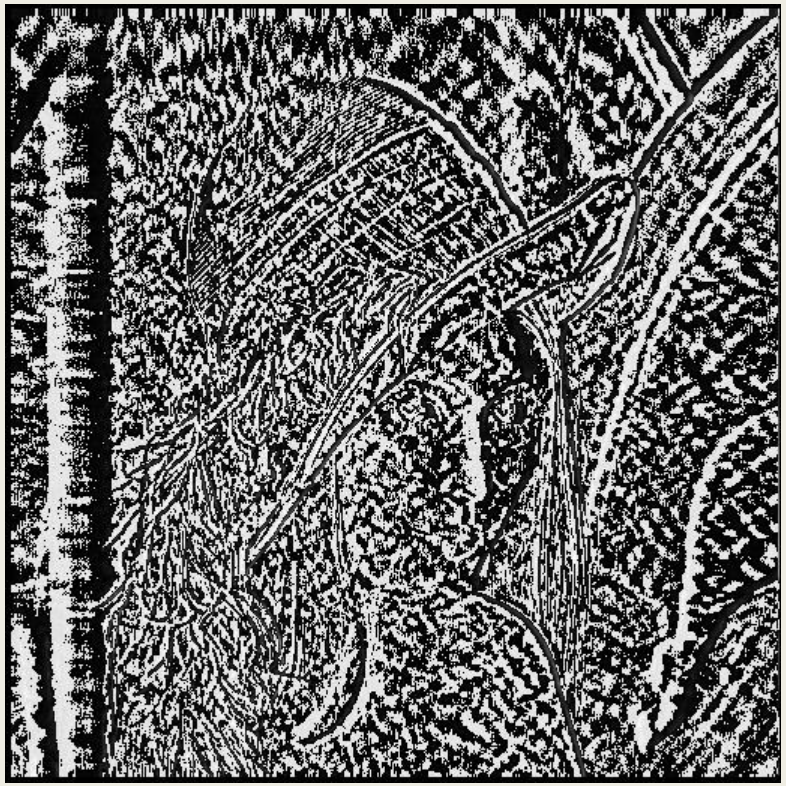
K = 400

K = 500

Conclusion(Error image from SVD and PGM)

Original image

SVD Error



PGM Error

Part 3: PRINCIPLE COMPONENT ANALYSIS (PCA)

What is PCA?

- Principal component analysis is a technique for feature extraction so it combines the input variables in a specific way, that we can drop the “least important” variables while still retaining the most valuable parts of all of the variables.
- As an added benefit, each of the “new” variables after PCA are all independent of one another. This is a benefit because the assumptions of a linear model require the variables to be independent of one another.
- PCA is typically used as an intermediate step in data analysis when the number of input variables is otherwise too large for useful analysis.

When should we use PCA?

- Do you want to reduce the number of variables, but aren’t able to identify variables to completely remove from consideration?
- Do you want to ensure your variables are independent of one another?
- Data Reduction : PCA is most commonly used to condense the information contained in a large number of original variables into a smaller set of new composite dimensions, with a minimum loss of information.
- Interpretation : PCA can be used to discover important features of a large data set. It often reveals relationships that were previously unsuspected, thereby allowing interpretations that would not ordinarily result.

How to Calculate PCs?

- Let G be the m*n matrix where m is the number of rows(samples) and n be the number of columns(attributes)
- Find the Mean

$$\overline{X} = \frac{\sum_{i=1}^{n} x_i}{n}$$

n = number of items in the sample

- Let B be a matrix whose column values are subtracted by column Mean

- Find Co- Variance Matrix S

$$S=B'B/(n-1)$$

S is a symmentric matrix here.

- Then find eigen values and eigen vectors of S.

$$S= VDV'$$

D is the diagonal matrix which contains eigen values of S which are arranged in decreasing order.

V is the matrix which contains eigen vectors of S.

V' is the transpose of V and it is also the rotational matrix.

- Principle Components is the product of attributes and V

Dataset and Implementation

- This dataset is about the crime reports for the states in USA from the years the years 1975-2015.
- It contains about 1354 rows and 8 attributes.
- The attributes are Report Year, State, Population, violent_crimes, homicides, rapes, assaults, robberies
- We used JMP to analyze the data. It is mainly used for statistical analysis.
- It gives you meaningful and simple solutions for the most toughest business problems.

Data Set

	A	B	C	D	E	F	G	H
1	Report_Year	State	Population	violent_crimes	homicides	rapes	assaults	robberies
2	1975	AZ	1121722	6726	100	470	3413	2743
3	1975	CA	6858823	67951	1166	3973	29555	33257
4	1975	CO	624796	5571	81	524	2227	2739
5	1975	DC	716000	12704	235	520	2812	9137
6	1975	FL	1922889	21778	371	916	11983	8508
7	1975	GA	490584	8033	185	443	3518	3887
8	1975	HI	705262	1596	58	169	319	1050
9	1975	IL	3150000	37160	818	1657	12514	22171
10	1975	IN	503411	4655	95	351	1117	3092
11	1975	KS	259240	1137	36	53	408	640
12	1975	KY	0	0	0	0	0	0
13	1975	LA	571934	5993	158	237	2002	3596
14	1975	MA	616120	11386	119	453	3036	7778
15	1975	MD	2571798	22459	349	945	8635	12530
16	1975	MI	1432444	30387	633	1424	7013	21317
17	1975	MN	405378	3326	47	307	1134	1838
18	1975	MO	1058094	16635	354	764	6148	9369
19	1975	NC	412043	2658	85	91	1439	1043
20	1975	NE	391389	2540	36	178	1115	1211
21	1975	NJ	372663	7136	122	297	2444	4273

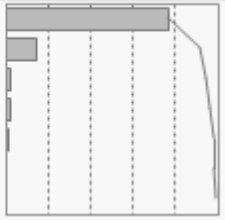
Co – Variance Matrix

- Co- Variance Matrix

	Report_Year	Population	violent_crimes	homicides	rapes	assaults	robberies
Report_Year	140.10355	1734922.0	-18980.85	-491.2751	-1201.506	2247.6213	-19561.88
Population	1734922.0	5.381e+12	5.634e+10	684708984	1.9414e+9	2.679e+10	2.692e+10
violent_crimes	-18980.85	5.634e+10	763913155	9316259.5	25023090	352510006	377119106
homicides	-491.2751	684708984	9316259.5	126265.32	328869.94	4194096.7	4668216.6
rapes	-1201.506	1.9414e+9	25023090	328869.94	1042112.5	11635433	12020831
assaults	2247.6213	2.679e+10	352510006	4194096.7	11635433	172181014	164532633
robberies	-19561.88	2.692e+10	377119106	4668216.6	12020831	164532633	195933133

Eigen Values and Eigen Vector

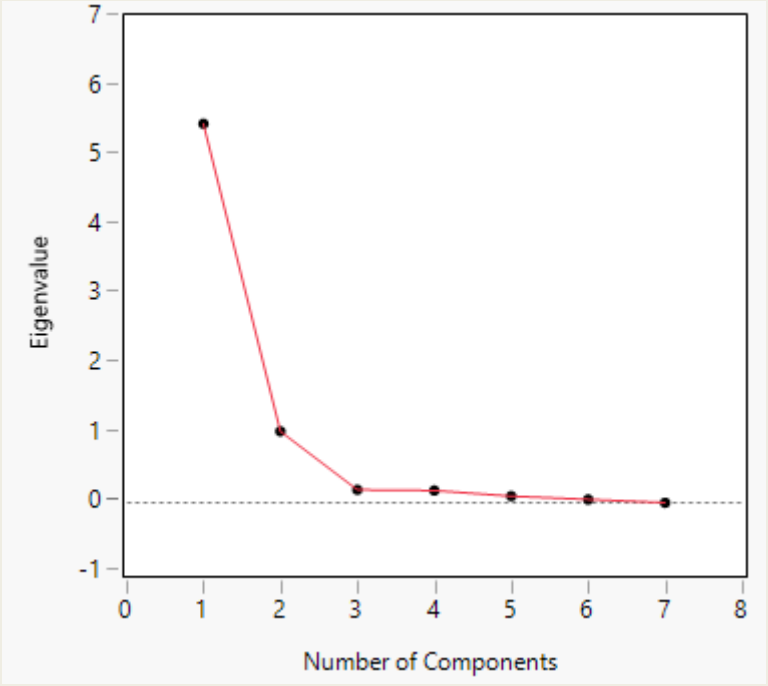
- Eigen Values

Number	Eigenvalue	Percent	20 40 60 80	Cum Percent
1	5.4645	78.064		78.064
2	1.0302	14.717		92.781
3	0.1863	2.662		95.443
4	0.1764	2.521		97.963
5	0.0954	1.362		99.326
6	0.0472	0.674		100.000
7	0.0000	0.000		100.000

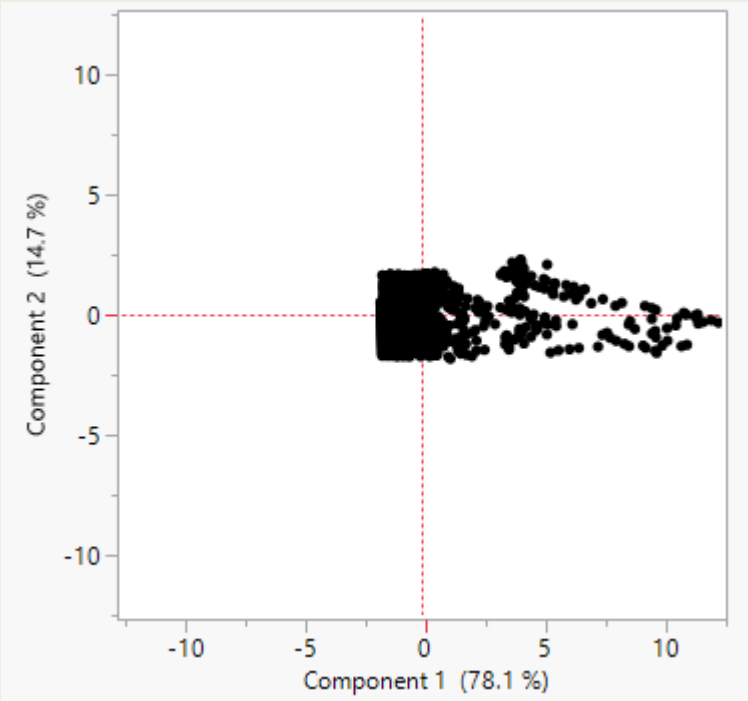
- Eigen Vectors

	Prin1	Prin2	Prin3	Prin4	Prin5	Prin6	Prin7
Report_Year	-0.02917	0.97959	0.15345	0.10401	0.06918	0.01987	0.00003
Population	0.39038	0.15089	-0.77165	-0.36850	0.30386	-0.03563	0.00015
violent_crimes	0.42342	0.00755	0.22450	-0.18888	-0.18762	0.15913	-0.82101
homicides	0.41346	-0.06376	0.27715	0.20746	0.36733	-0.75505	0.01042
rapes	0.39791	-0.04424	-0.22568	0.81600	-0.03197	0.34786	0.03010
assaults	0.41211	0.08748	0.00257	-0.15626	-0.77921	-0.19742	0.38980
robberies	0.41031	-0.06242	0.45041	-0.29053	0.35292	0.49289	0.41592

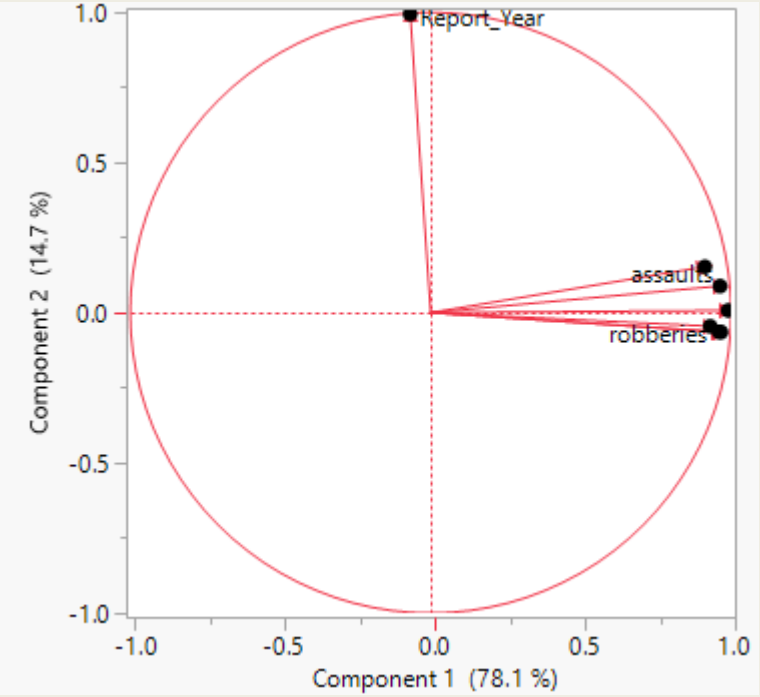
Screen Plot



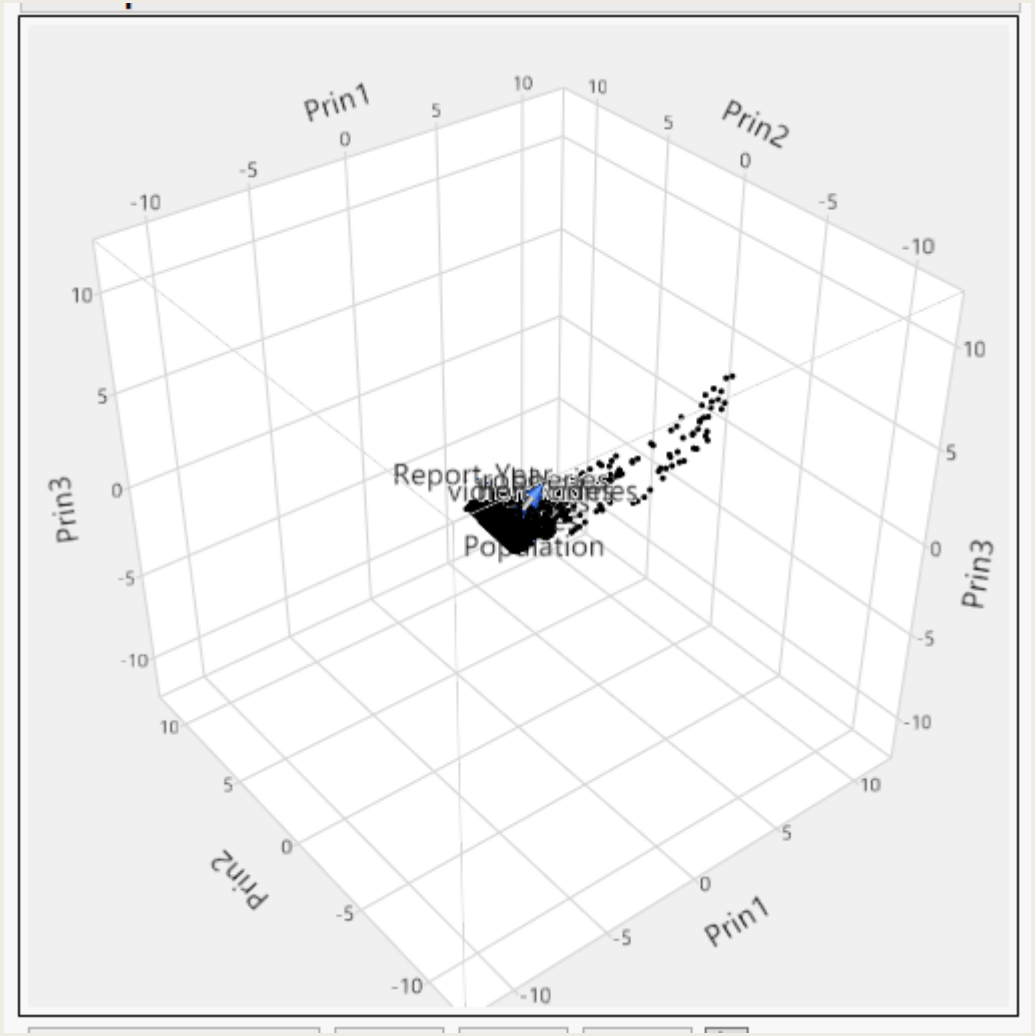
Score Plot



Loading Plot



Scatter Plot 3D



Questions!

References

- https://pdfs.semanticscholar.org/c1a7/ece080556ece54025a74b5889b36b7d8c36e.pdf
- https://plot.ly/create/
- http://web.cs.iastate.edu/~smkautz/cs227f11/examples/week11/PGMUtils.java
- http://web.cs.iastate.edu/~smkautz/cs227f11/examples/week11/pgm_files.pdf
- https://www.youtube.com/watch?v=c0YM8Vvf7iY
- https://stackoverflow.com/questions/18247766/java-splitting-integer-into-2-bytes-and-then-combining-them-again-into-an-integ
- http://www.javapractices.com/topic/TopicAction.do?Id=245