MA515 - Foundation of Data Science

Project Report

Members:

Prashant(2020CSB1113)
Prachi(2020EEB1048)
Anubhav(2020CSB1073)
Shyam(2020CSB1110)

Problem statement - 1:

Use Decision Tree, Random Forests, KNN and LDA to predict whether the person will default. Compare the findings from different methods.

Exploratory Data Analysis:

In Default.xlsx dataset columns present were:

- An unnamed column
- Default
- Student
- Balance
- Income.

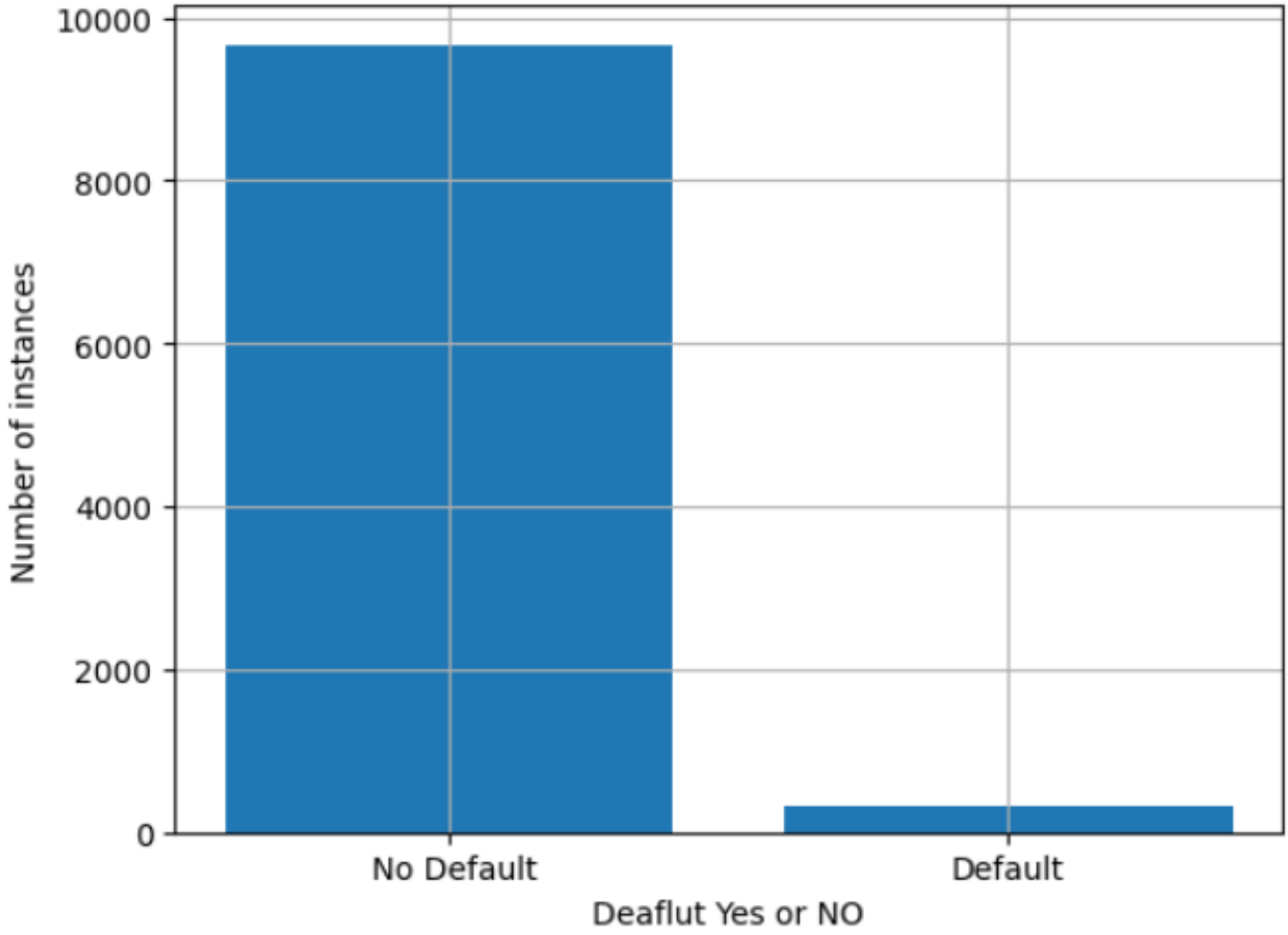
	balance	income
count	10000.000000	10000.000000
mean	835.374886	33516.981876
std	483.714985	13336.639563
min	0.000000	771.967729
25%	481.731105	21340.462903
50%	823.636973	34552.644802
75%	1166.308386	43807.729272
max	2654.322576	73554.233495

- Unnamed Column is not needed in our model training hence dropped.
- Default is our dependent variable and its attributes are Student, Balance and Income.
- Default is a categorical variable with two categories (Yes and No).
- Student is also a categorical variable(Yes and No).

- Balance and Income are continuous numerical variables.
- We then check the statistical information like mean, frequency,standard deviation,median etc for numerical data (this is shown by ‘data.describe()’).

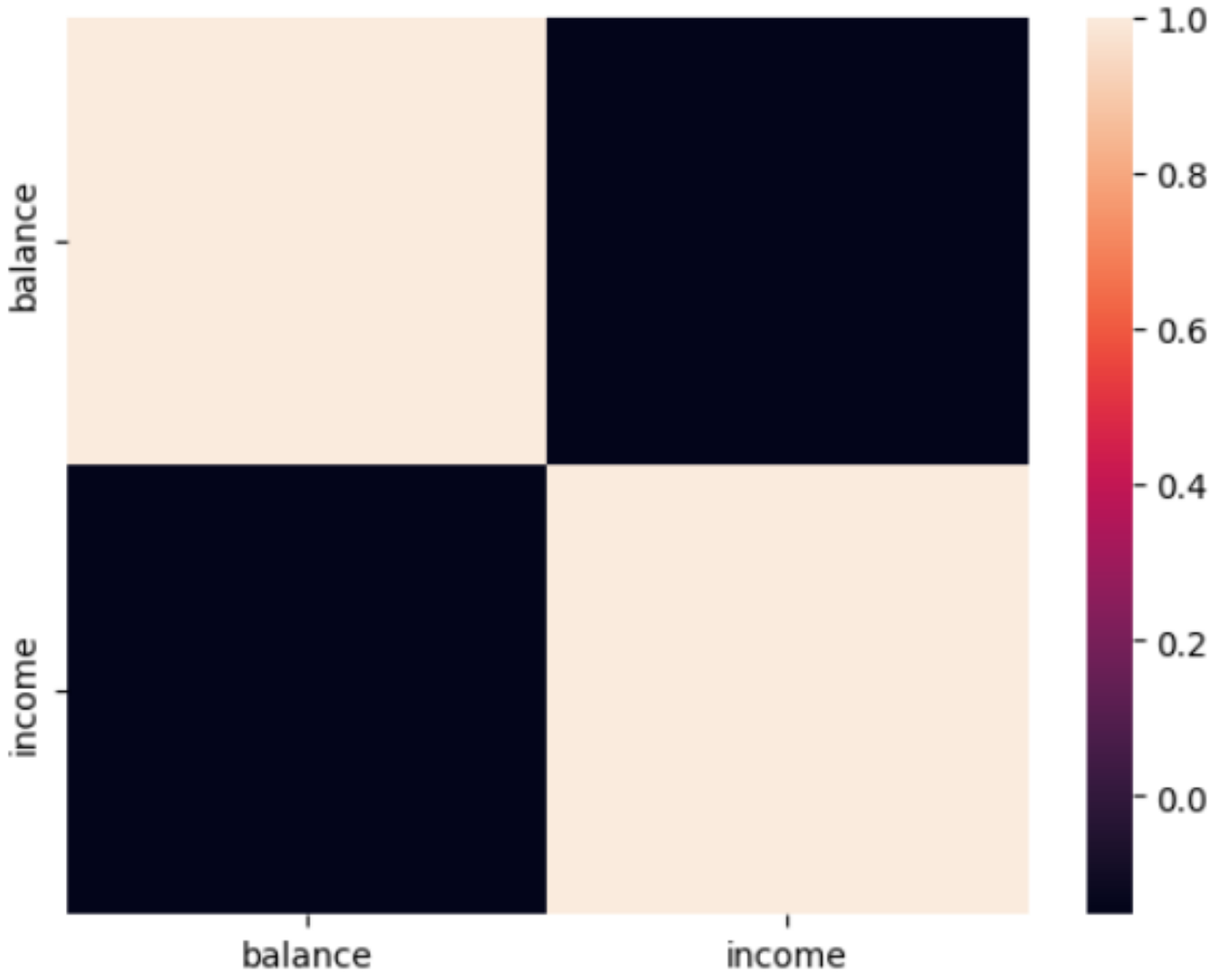
For categorical data we checked the instances of Default ‘Yes’ and Default ‘No’. We can observe here that instances where card defaulted were rare but a good model should be able to predict accurately when the card defaults.

```
The number of instances for No (0): 9667
The number of instances for Yes (1) : 333
```

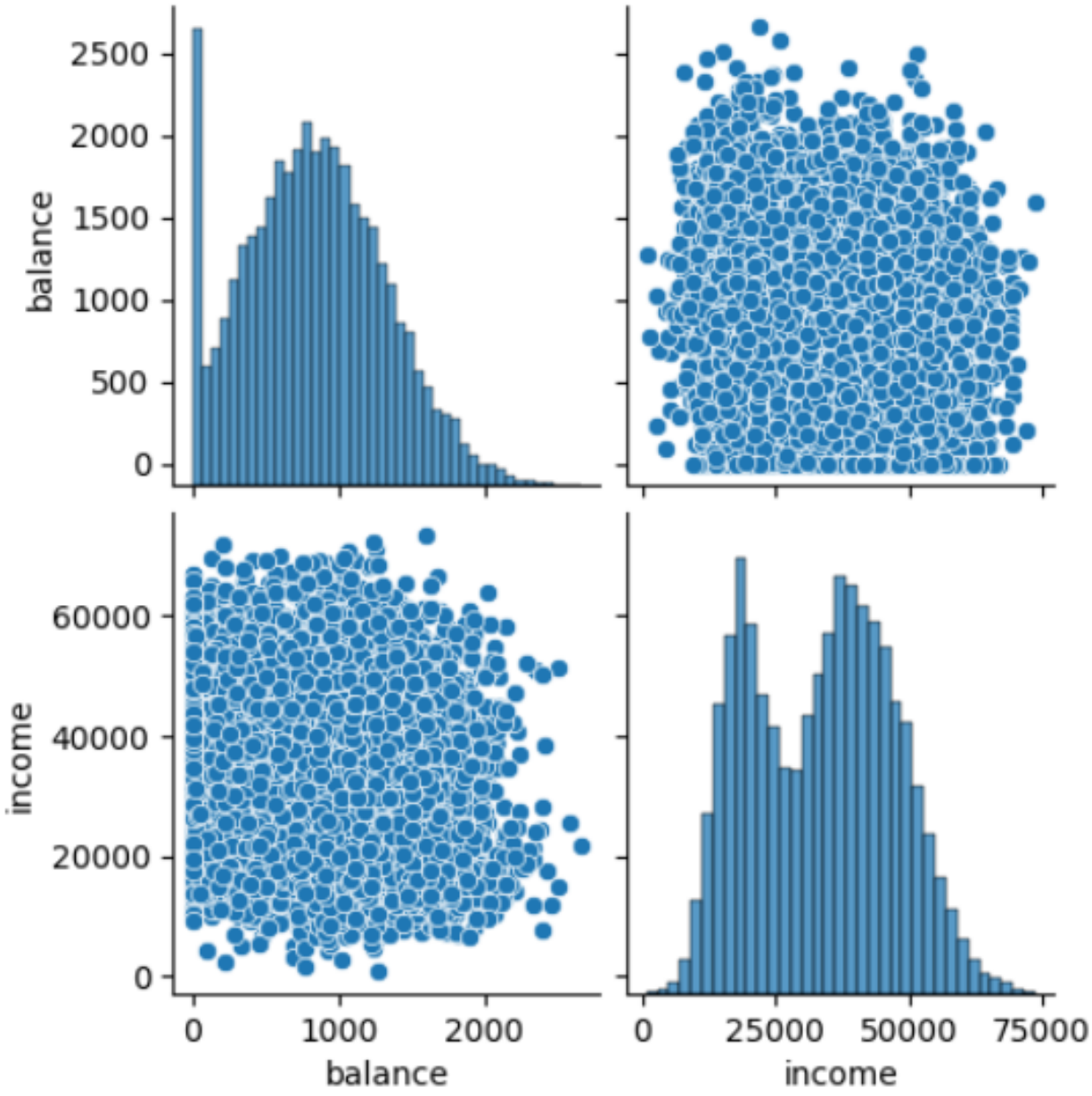


Check the correlation of Balance and Income. This is shown as a heatmap.

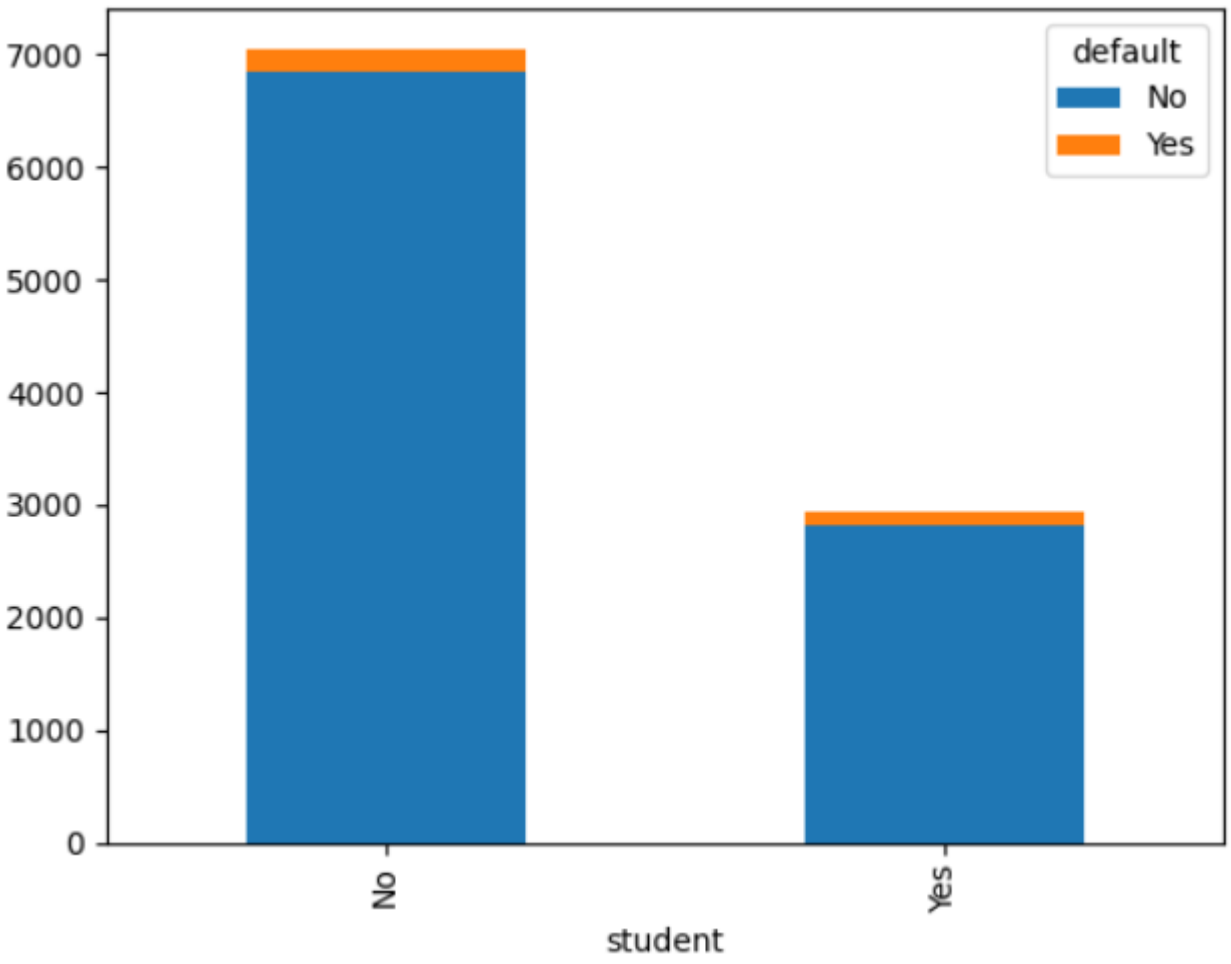
There is no steady relation between balance and income.



Correlation heatmap

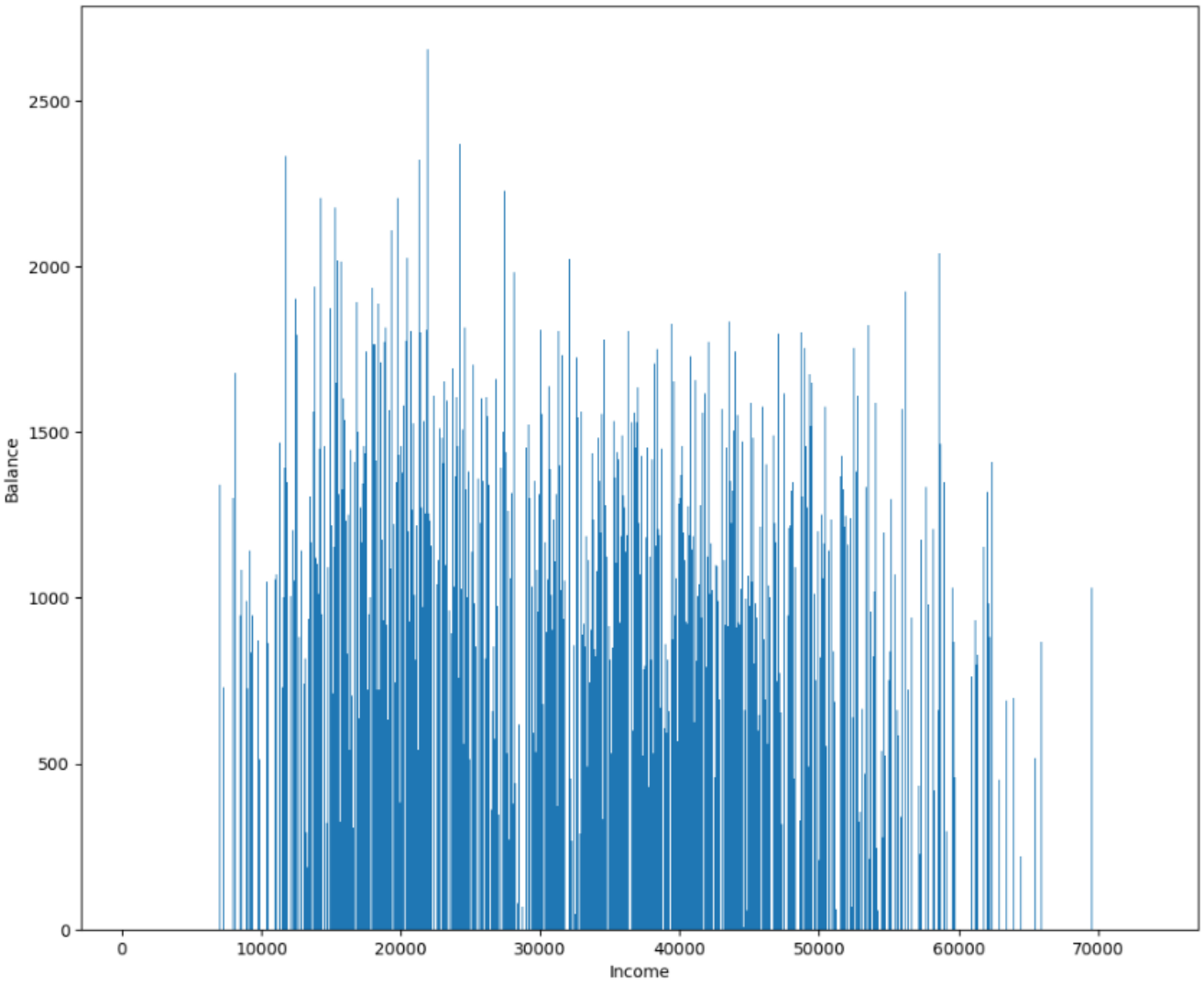

2500
2000
1500
1000
500
0
balance
60000
40000
20000
0
income
0
1000
2000
balance
0
25000
50000
75000
income

Pairplot



student	default	
No	No	6850
	Yes	206
Yes	No	2817
	Yes	127

Relation between student and default



Balance and income relation in the form of a bar graph

Scaled the data so that computation becomes easier using Standardization.

Linear Discriminant Analysis (LDA)

Creates a classifier axis between data points to predict classes. To create an axis, Linear Discriminant Analysis uses the following criteria:

- It maximizes the distance between means of two classes.
- It minimizes the variance within the individual class.

Trained the dataset on LDA. X_train and y_train are used for learning.

We achieved accuracies as:
LDA Training Accuracy: 97.26666666666667%
LDA Testing Accuracy: 97.04%

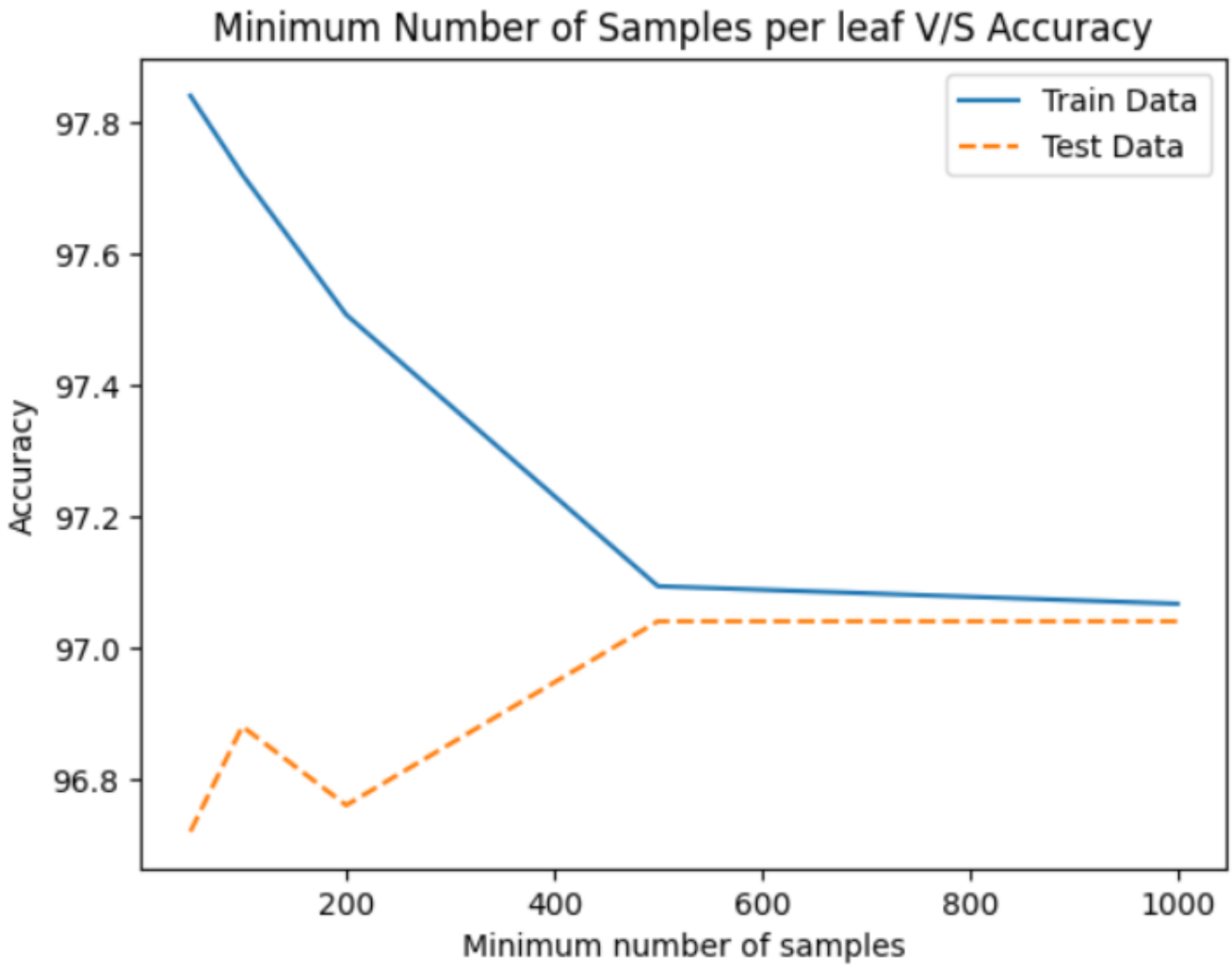
The accuracy of the model is also verified by K-Fold Validation to get a better understanding of accuracy and to reduce bias. The resulting accuracy is also near 97.3%. K=10 is taken.

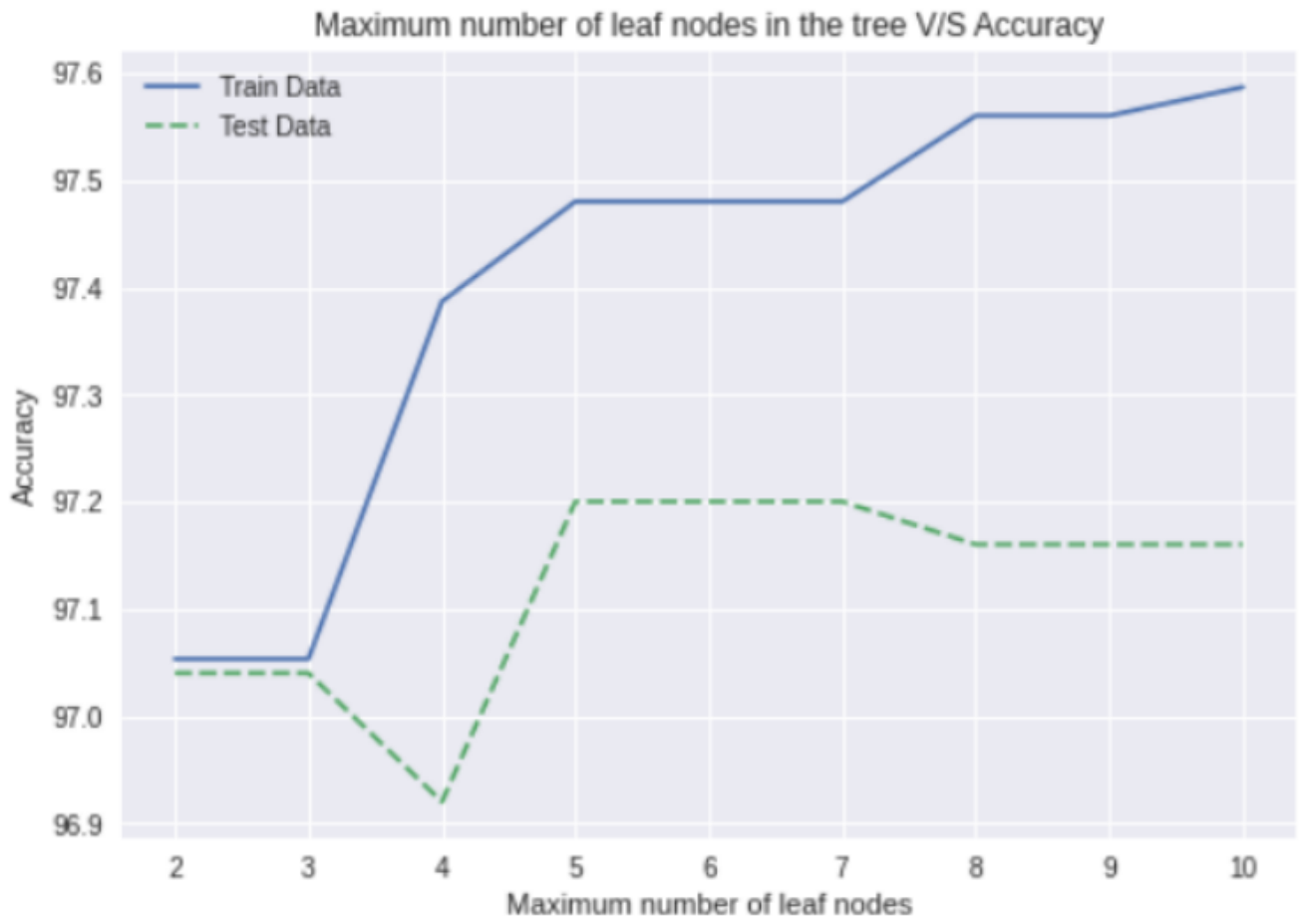
Fold: 1, Training/Test Split Distribution: [6536 214], Accuracy: 0.976
Fold: 2, Training/Test Split Distribution: [6536 214], Accuracy: 0.975
Fold: 3, Training/Test Split Distribution: [6536 214], Accuracy: 0.976
Fold: 4, Training/Test Split Distribution: [6537 213], Accuracy: 0.967
Fold: 5, Training/Test Split Distribution: [6537 213], Accuracy: 0.980
Fold: 6, Training/Test Split Distribution: [6537 213], Accuracy: 0.972
Fold: 7, Training/Test Split Distribution: [6537 213], Accuracy: 0.971
Fold: 8, Training/Test Split Distribution: [6537 213], Accuracy: 0.968
Fold: 9, Training/Test Split Distribution: [6537 213], Accuracy: 0.971
Fold: 10, Training/Test Split Distribution: [6537 213], Accuracy: 0.972

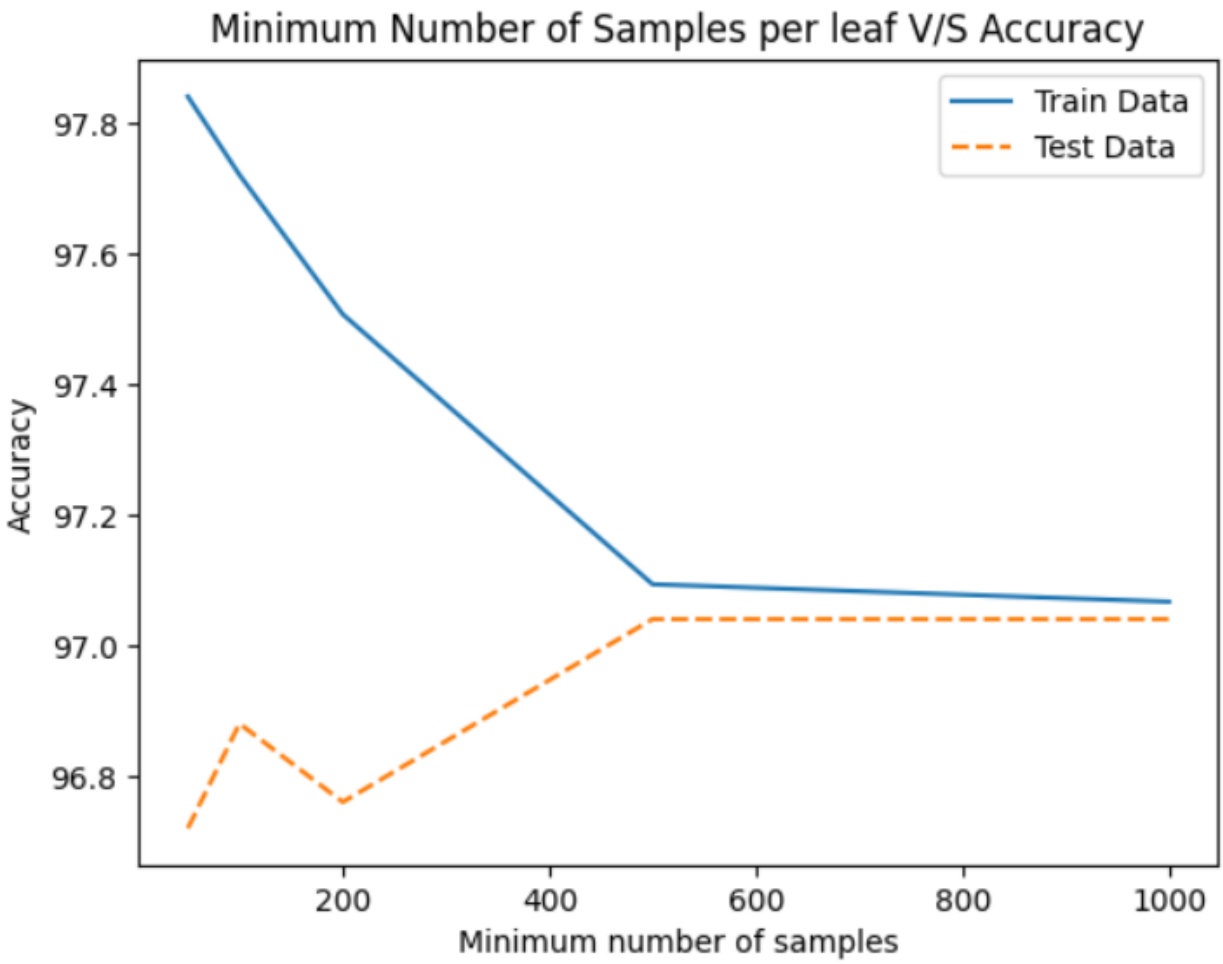
Cross-Validation accuracy: 0.973 +/- 0.004

Decision Tree for Classification

Observations from experiments done.
1. We can see that when min_sapmle_split is 200 the test accuracy is really low but training accuracy is high. Increasing min_sample_split they both converge near 97.
2. Ideal min_sample_split will be 800 as here both training and testing accuracy converges.
3. As Maximum number of leaf Nodes increases both testing and training accuracy increases
4. We take 10 as ideal Maximum Number of leaf Nodes
5. As Max tree depth increases Accuracy of both testing and training increases




Maximum number of leaf nodes in the tree V/S Accuracy
Train Data
Test Data
Accuracy
97.6
97.5
97.4
97.3
97.2
97.1
97.0
96.9
2
3
4
5
6
7
8
9
10
Maximum number of leaf nodes



Now we have our ideal criteria to train a decision tree.

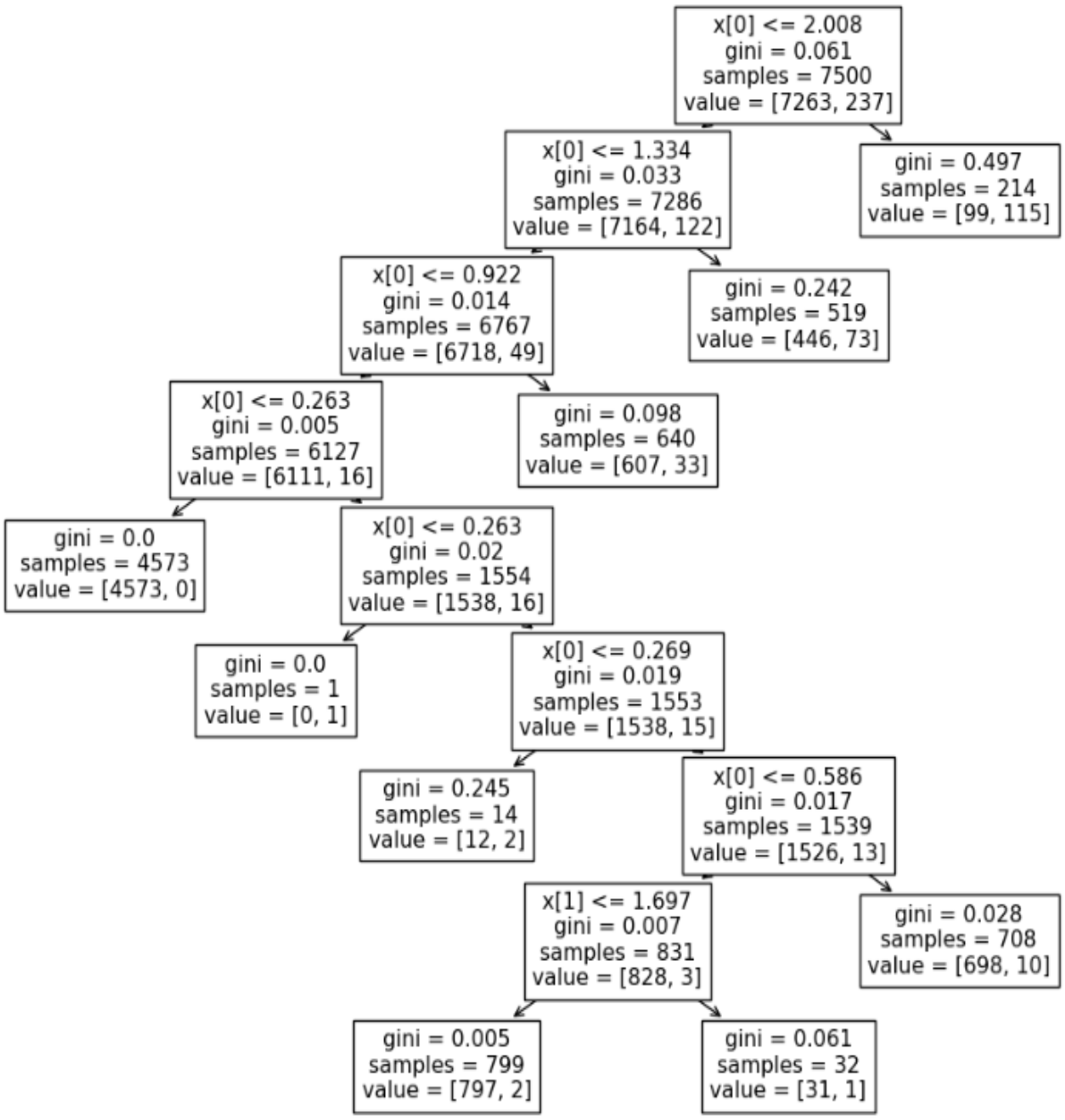
We test the learned model over a testing dataset. Accuracy Score is as follow:

- Confusion Matrix for test accuracy
  [2386 18]
  [56 40]

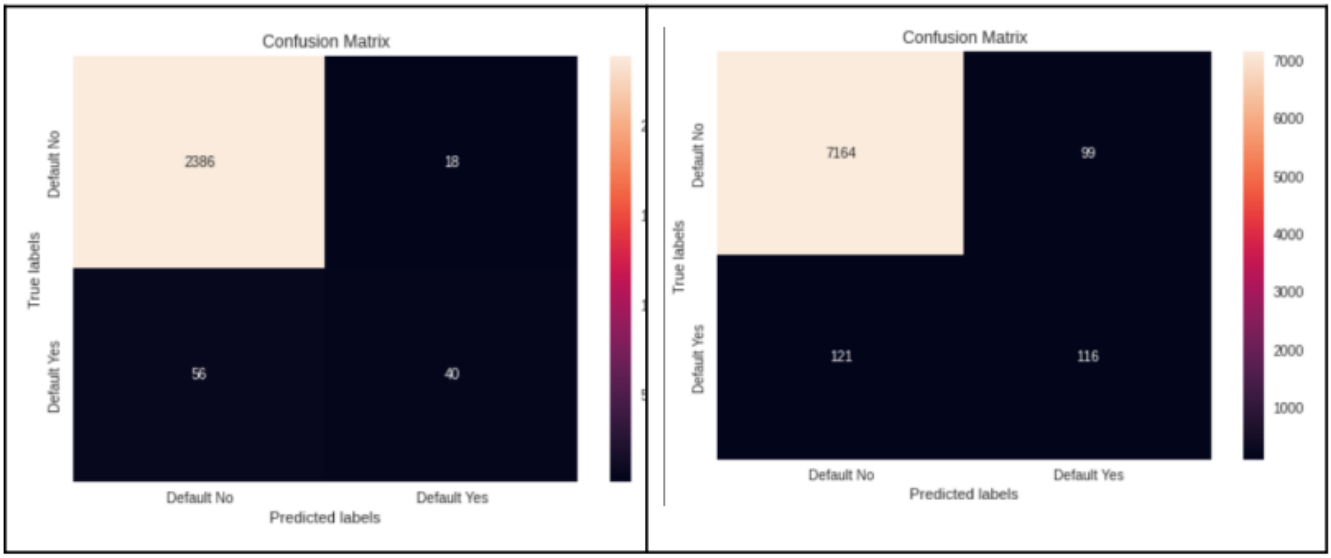
- Testing Classifier Score 0.9704

- Confusion Matrix for training data
  [7164 99]
  [121 116]

- Training Classifier Score 0.9706



DECISION TREE VISUALIZATION



Testing Confusion Matrix and Training Confusion Matrix

The accuracy of the model is also verified by K-Fold Validation to get a better understanding of accuracy and to reduce bias. The resulting accuracy is also near 97%. K=10 is taken

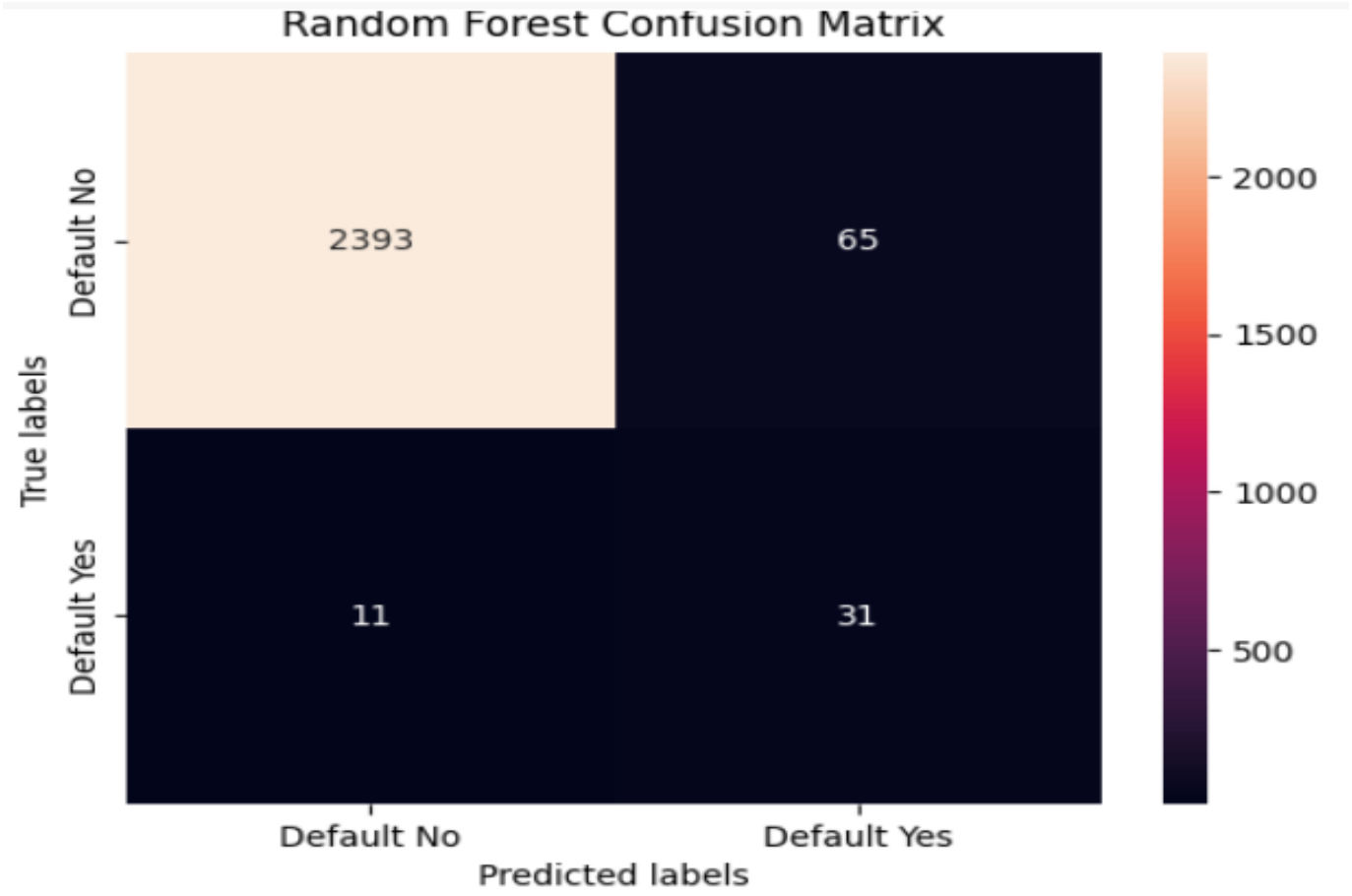
```
account_circle
Fold:  1, Training/Test Split Distribution: [6536  214], Accuracy: 0.972
Fold:  2, Training/Test Split Distribution: [6536  214], Accuracy: 0.965
Fold:  3, Training/Test Split Distribution: [6536  214], Accuracy: 0.976
Fold:  4, Training/Test Split Distribution: [6537  213], Accuracy: 0.960
Fold:  5, Training/Test Split Distribution: [6537  213], Accuracy: 0.972
Fold:  6, Training/Test Split Distribution: [6537  213], Accuracy: 0.972
Fold:  7, Training/Test Split Distribution: [6537  213], Accuracy: 0.976
Fold:  8, Training/Test Split Distribution: [6537  213], Accuracy: 0.971
Fold:  9, Training/Test Split Distribution: [6537  213], Accuracy: 0.968
Fold: 10, Training/Test Split Distribution: [6537  213], Accuracy: 0.971


Cross-Validation accuracy: 0.970 +/- 0.005
```

Random forest

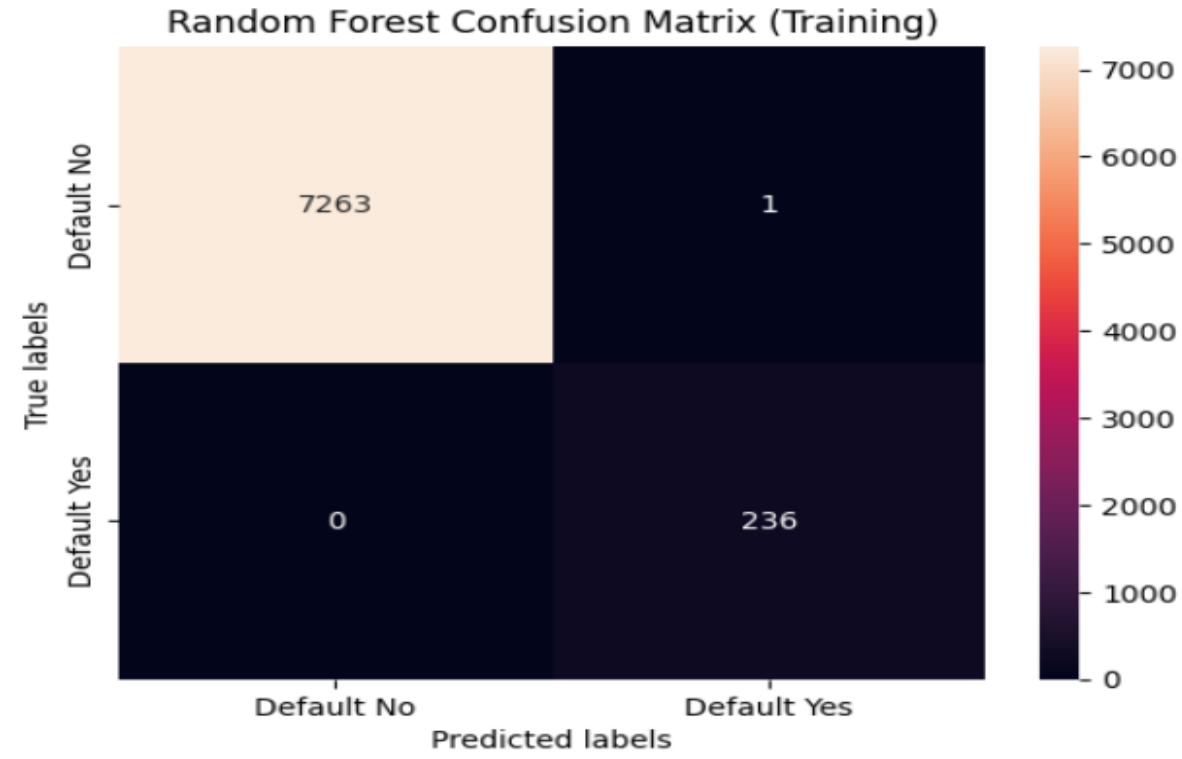
Overview:
The Random Forest model demonstrates robust performance across various evaluation metrics, showcasing its efficacy in predictive tasks. Below is an analysis of the model's performance based on key metrics.

Model Evaluation Metrics:

Test Accuracy: 96.96%



Training Accuracy: 99.98%




```
Random Forest Training Accuracy: 99.98666666666666 %
```

The model achieves exceptionally high accuracy of nearly 100% on the training dataset, indicating its ability to fit the training data almost perfectly.

Error Rate (Test Data, Default 'Yes'): 26.19%

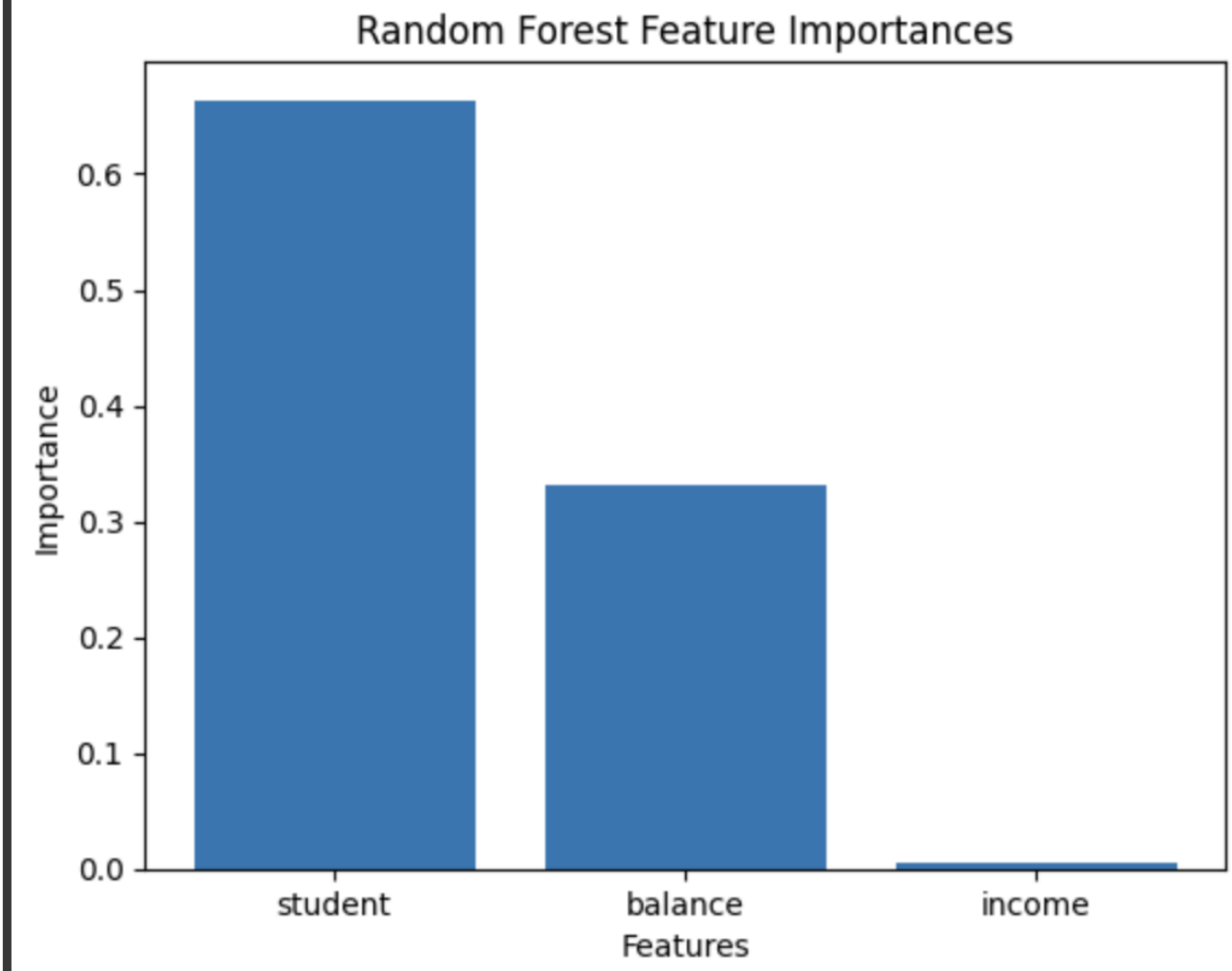
The error rate on the test data for the positive class ('Yes') stands at around 26%, suggesting a moderate misclassification rate for the positive class.

Error Rate (Train Data, Default 'Yes'): 0.0%

The model achieves zero error rate on the training data for the positive class ('Yes'), signifying perfect classification on the training set.

K-Fold Cross-Validation Accuracy: 96.9% (+- 0.5)

```
Random Forest Cross-Validation accuracy: 0.969 +/- 0.005
```



The K-fold cross-validation accuracy indicates the model's stability, showcasing an average accuracy of 96.9% with a low variance of approximately 0.5%.

AUC Score: 1.0

The AUC (Area Under the Curve) score of 1.0 signifies outstanding performance in distinguishing between positive and negative classes, demonstrating excellent predictive power without misclassification.
Insights and Analysis:

High Accuracy:

The model demonstrates commendable accuracy on both training and test datasets, indicating its robustness in making accurate predictions.

Conclusion:

The Random Forest model demonstrates impressive performance metrics, exhibiting high accuracy, low error rates, and a perfect AUC score. However, careful consideration should be given to potential overfitting and class imbalance issues. Further optimization, regularization, or addressing class imbalance might enhance the model's generalization capabilities.

This report summarizes the Random Forest model's performance based on the provided evaluation metrics, highlighting its strengths and potential areas for improvement.

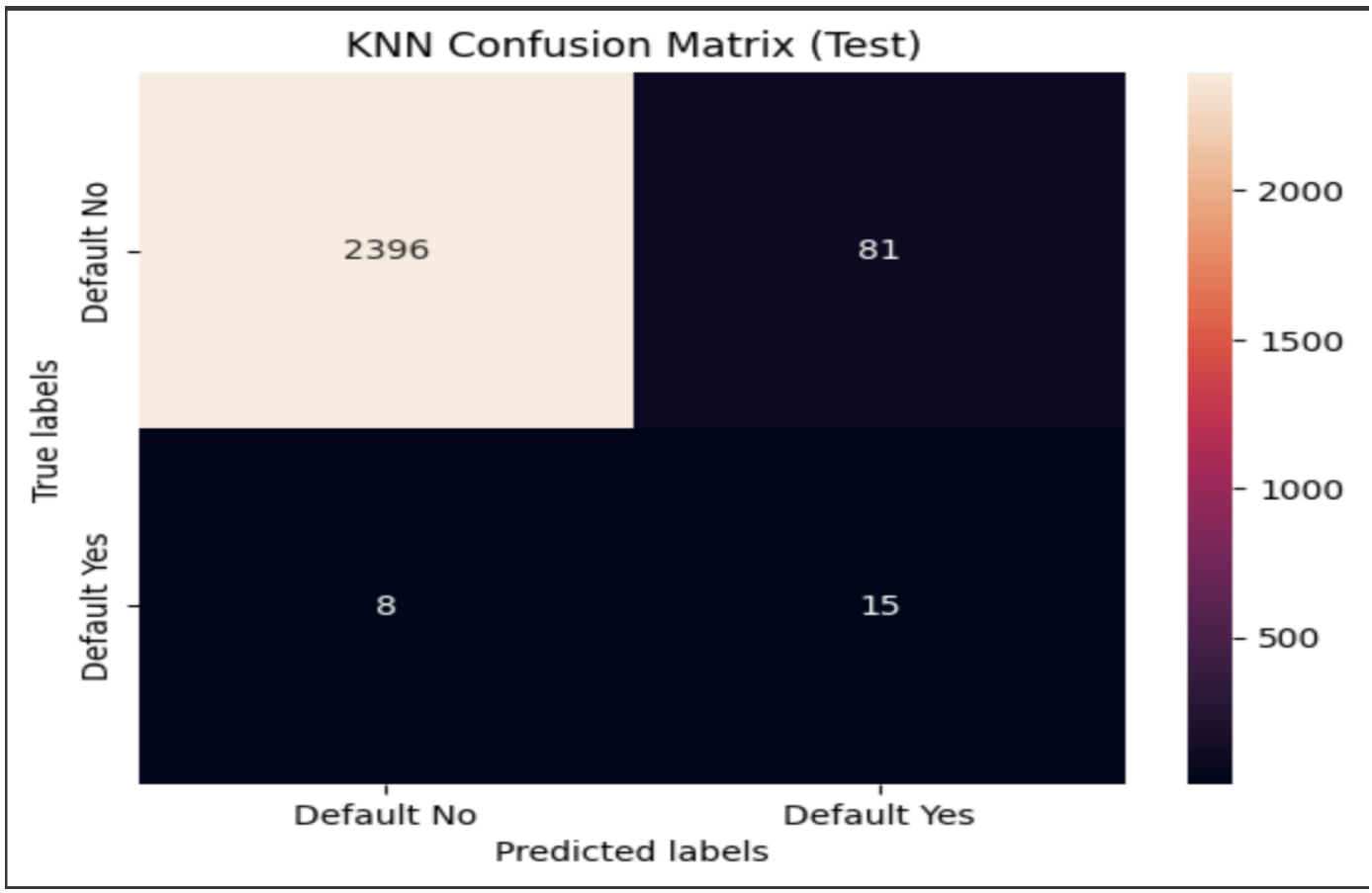
K-Nearest Neighbors (KNN):

Overview:

The K-Nearest Neighbors model demonstrates a strong performance across various evaluation metrics, indicating its effectiveness in predictive tasks. Below is an analysis of the model's performance based on key metrics.

Model Evaluation Metrics:

1. Test Accuracy: 97.61%
- The model showcases high accuracy, achieving an accuracy of approximately 98% on the test dataset.



2. Training Accuracy: 96.44%

- The model achieves a commendable accuracy of approximately 96% on the training dataset, suggesting its ability to fit the training data quite well.

3. Error Rate (Test Data, Default 'Yes'): 2.259%

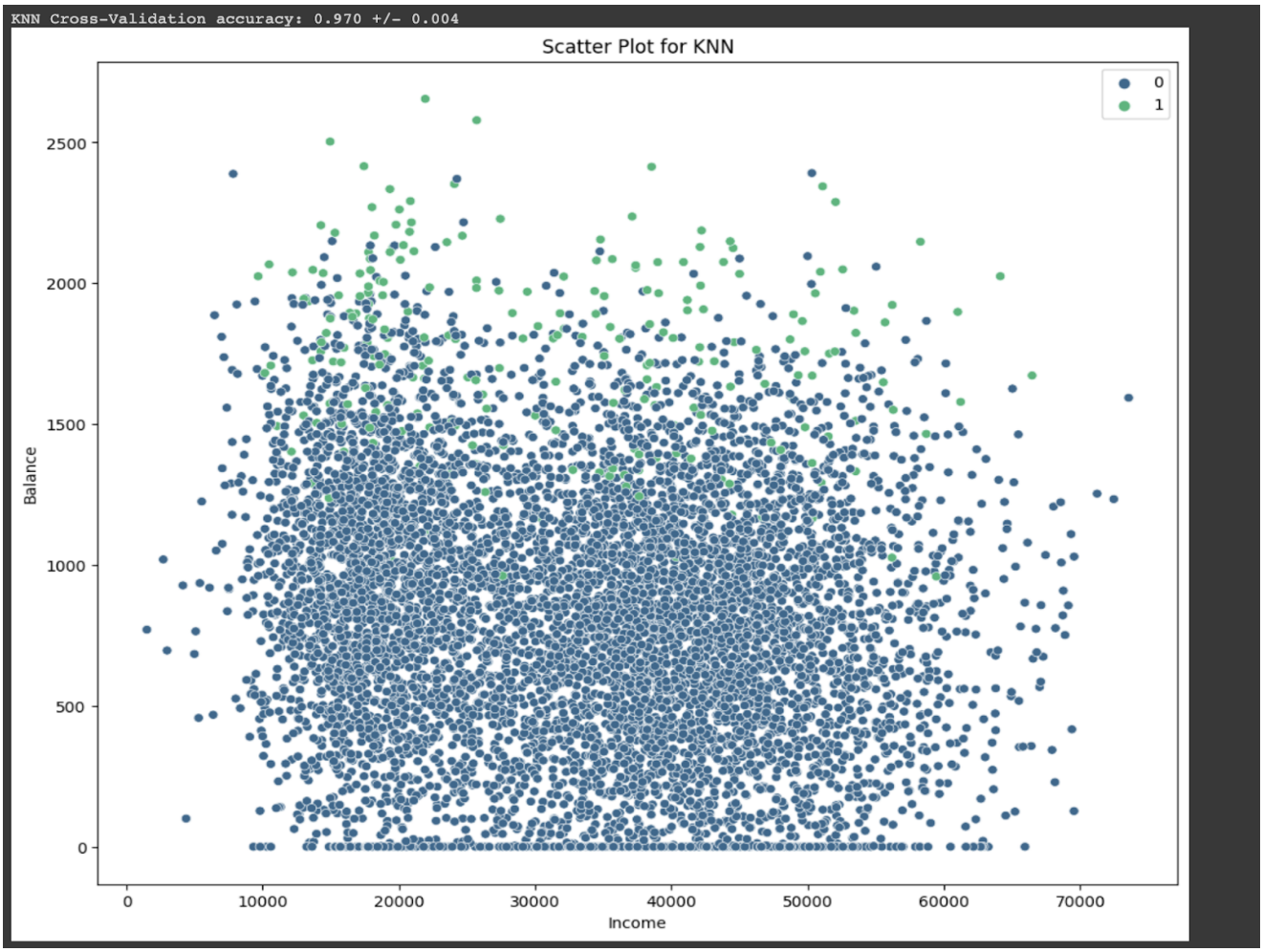
- The error rate on the test data for the positive class ('Yes') stands at approximately 2.26%, indicating a low misclassification rate for the positive class.

4. Error Rate (Train Data, Default 'Yes'): 0.60%

- The model achieves an error rate of around 0.60% on the training data for the positive class ('Yes'), indicating a low misclassification rate on the training set.

5. K-Fold Cross-Validation Accuracy: 97.0% (+- 0.4)

- The K-fold cross-validation accuracy reveals an average accuracy of 97.0% with a low variance of approximately 0.4%, showcasing stability and consistent performance across folds.



6. AUC Score: 0.991

- The AUC (Area Under the Curve) score of 0.991 suggests excellent discriminatory ability, depicting strong performance in distinguishing between positive and negative classes.

Insights and Analysis:

- High Accuracy:
The model demonstrates impressive accuracy on both training and test datasets, indicating its ability to make accurate predictions.

- Generalization:
There is a slight difference between training and test accuracies, suggesting moderate generalization capability. However, the performance on unseen data remains quite high.

- Low Error Rates:
The low error rates on both training and test datasets for the positive class ('Yes') indicate the model's effectiveness in correctly classifying positive instances.

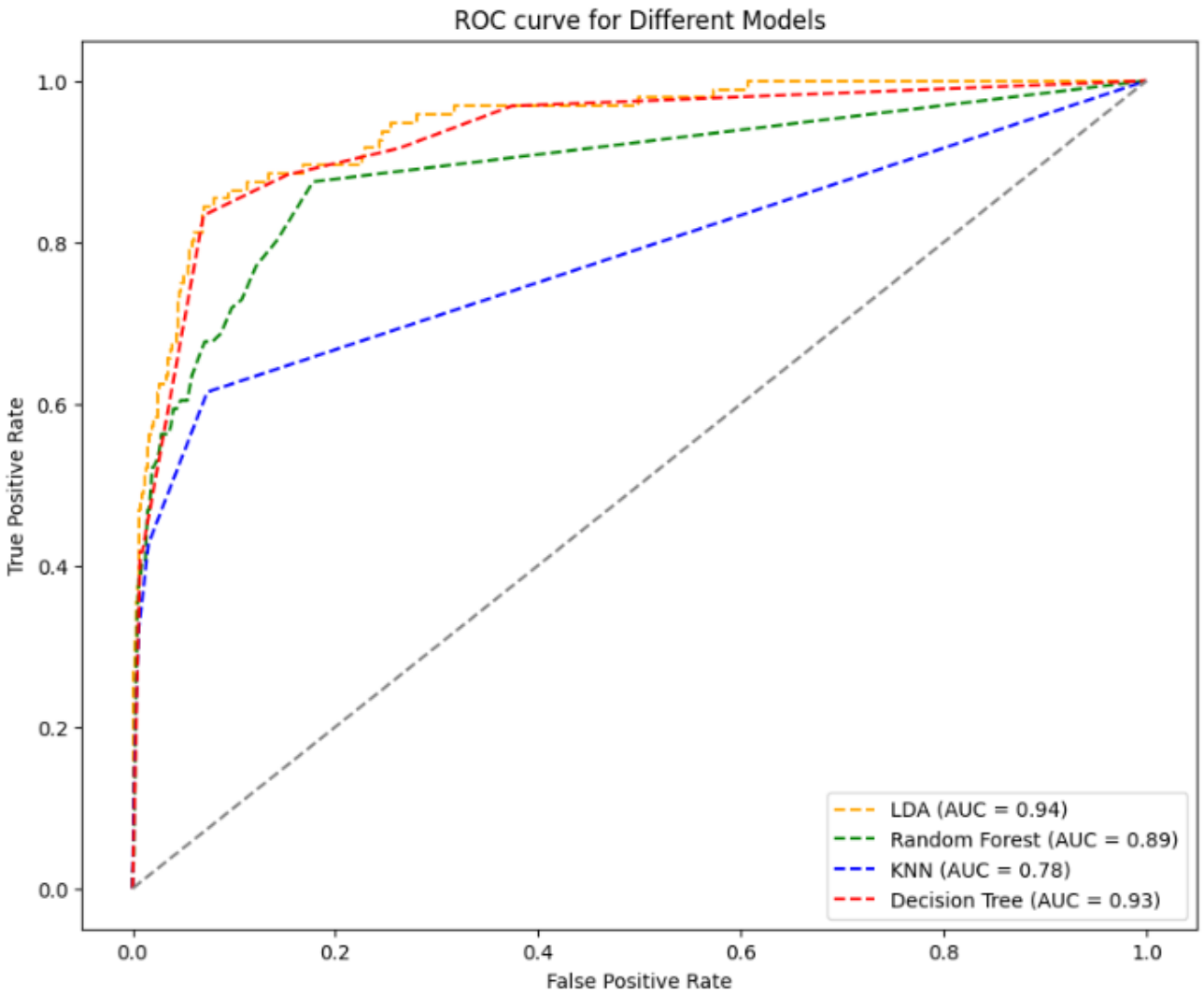
Conclusion:

The K-Nearest Neighbors (KNN) model exhibits strong performance across various metrics, showcasing high accuracy, low error rates, and a high AUC score. It demonstrates good generalization to unseen data while maintaining strong discriminatory ability between classes.

Comparing Final Results

Model	Test Accuracy	Training Accuracy	Error Rate (Test Data, Default 'Yes')	Error Rate (Train Data, Default 'Yes')	K-Fold Cross Validation Accuracy	AUC Score
Linear Discriminant Analysis	97.04%	97.27%	7.69%	2.65%	97.3% (+-0.4%)	0.982
Decision Tree	97.04%	97.07%	58.83%	51.05%	97.0% (+-0.5%)	0.724
Random Forest	96.96%	99.98%	26.19%	0.0%	0.969 (+-0.005)	1.0
KNN	97.61%	96.44%	2.259%	0.60%	0.970 (+-0.004)	0.991

Plot ROC curves



Hence from Comparison Random Forest is a better choice over all.

Problem statement - 2:

Use PCA and SVD to reduce the dimensions of the data 2 so that the reduced data has 5, 10, 15 and 20 columns. Give an exploratory data analysis on SVS with 5 columns.

Data:

Unnamed: 0	0	1	2	3	4	5	6	7	8	9	...	41	42	43	44	45	46	47	48	49	50
0	0	40.527005	23.827908	70.151381	25.402995	51.205610	-0.500565	14.055124	34.628477	9.124017	...	18.064201	9.432148	2.504029	0.269358	19.952375	7.455010	9.459746	39.861311	6.326133	20.627744
1	1	16.131192	3.417445	-4.161446	44.970951	9.048227	1.652619	6.938698	15.127169	-6.665835	...	24.769010	101.715152	-1.039992	18.389691	4.058942	9.340447	8.972296	7.967516	5.644450	31.050950
2	2	31.711781	1.921228	34.673047	50.258594	80.429873	0.171150	16.645188	33.591276	10.008848	...	24.733085	12.553875	13.120951	34.285047	41.687390	50.279687	24.313796	4.485158	-8.039312	30.973228
3	3	22.344608	17.654925	-7.462117	11.181247	9.024700	43.636999	5.731934	62.434215	34.540465	...	13.562030	10.719871	0.473489	1.153721	14.223376	28.148071	-3.480725	49.718266	36.347201	8.920897
4	4	1.463305	70.867143	26.463260	35.616958	11.806956	35.414390	22.295906	27.546526	10.187778	...	13.428348	45.668807	23.522061	56.072090	3.727361	7.822075	19.567931	13.023900	10.670013	17.301297
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2495	2495	-1.141026	-1.017415	10.463158	2.887700	17.035051	4.059348	49.155453	9.383939	11.090642	...	3.276487	15.801198	86.279789	27.326991	18.040121	22.151473	75.854007	37.193650	29.466767	24.558372
2496	2496	58.287081	20.095938	2.865355	52.423177	-1.039915	17.328451	6.973377	30.851782	19.193449	...	-1.027688	51.109309	58.342683	27.669227	38.860496	73.064009	35.407761	38.432561	41.889405	15.256207
2497	2497	-3.548954	16.754013	12.868025	42.600501	10.188366	9.905352	2.521764	-2.002795	1.538263	...	23.128603	-3.803400	21.272966	40.937706	16.283572	43.449431	11.659081	86.987480	16.044745	5.368698
2498	2498	53.577562	20.097276	40.282856	8.647917	16.646555	19.588062	61.227117	10.292142	29.436988	...	24.711134	36.485876	9.940823	7.379581	14.054125	17.712843	-3.330291	50.057037	8.968517	32.313652
2499	2499	27.049408	7.280802	43.067183	6.048532	61.661639	8.190058	22.041664	15.934858	4.444227	...	18.668000	10.350884	48.764519	-1.566271	24.815107	0.416476	3.837631	7.054202	82.843595	41.563674

2500 rows × 51 columns

The dataset used is HD Data 2.csv. It consists of 2500 rows and 51 columns.

Results:

PCA -

PCA with different number of columns

```
PCA with 5 columns:
[[ 1249.6354379      2.5307997     12.82818027    -5.43261838
     37.82046779]
 [ 1248.61433583     4.40813143    14.63784728   -15.13313436
      5.83000866]
 [ 1247.6250026    -19.84826677   -22.14283521    33.94984933
      6.26145547]
 ...
 [-1247.50660175    10.6093287    -38.13364786   -31.77500735
      8.59765852]
 [-1248.47497842   -16.33285997    -6.97820136    45.33607387
     25.00529124]
 [-1249.51589957    38.88629505    20.30067045    37.21780314
    -36.34502967]]


PCA with 10 columns:
[[ 1.24963544e+03  8.72360102e+00 -1.07229191e+01 ...  3.89206026e-01
   1.75501149e+01  7.90771030e+00]
 [ 1.24861434e+03 -9.50699079e+00 -6.90983589e-01 ...  1.06662766e+01
   1.16410487e+01 -7.22478546e+00]
 [ 1.24762500e+03  3.52483272e+01 -2.78605989e+01 ... -2.00279044e+00
   3.28072540e+01  2.57572501e+01]
 ...
 [-1.24750660e+03  2.37553981e+00 -1.89438987e+01 ...  1.79432883e+01
  -7.16032649e+00 -2.74759671e+01]
 [-1.24847498e+03  1.03435649e+01  2.36542484e+00 ... -1.57029133e+00
   2.06217844e+01 -1.22224817e+01]
 [-1.24951590e+03 -3.40850983e+01  1.89413443e+00 ... -8.81014992e+00
   5.01758508e+01  5.21465778e+00]]


PCA with 15 columns:
[[ 1249.6354379      8.12887669     5.33996012 ...    34.58512998
    -38.17285551    34.53763401]
 [ 1248.61433583    23.28627131    -3.75040251 ...    31.16552479
      7.40814666     7.80160832]
 [ 1247.6250026    -43.82212856    13.61533471 ...    -8.07594913
    -13.06620764    35.60621881]
 ...
 [-1247.50660175    10.42521041    40.73705521 ...    18.37659358
    -13.95372369   -24.83497912]
 [-1248.47497842   -19.4666031     -9.77571765 ...    20.76226267
     26.21868694   -22.477989  ]
 [-1249.51589957    15.67932254   -11.94704016 ...   -14.80255862
    -26.3383094     20.17706723]]


PCA with 20 columns:
[[ 1.24963544e+03  2.22374502e+00 -5.81900503e+00 ...  8.98732815e+00
  -1.74177188e+01  2.99783560e+01]
 [ 1.24861434e+03 -5.98206583e+00 -8.99046135e+00 ... -8.49038497e+00
   1.91093349e+01  4.19065908e+01]
 [ 1.24762500e+03  4.40063805e+01  1.40866681e+01 ...  1.65660858e-01
   7.80274687e+00  7.76058073e-01]
 ...
 [-1.24750660e+03 -8.67623531e+00  2.31532564e+01 ... -9.54723956e-01
   3.41348031e+01 -2.86175802e+01]
```

SVD -

SVD on different number of columns


```
{5: array([[-1.07620749e-01,  2.24447056e+01,  2.10528209e+01, ...,
        1.90015237e+01,  2.23188490e+01,  1.96076932e+01],
      [ 9.28684513e-01,  2.32577490e+01,  2.09330989e+01, ...,
        1.90570959e+01,  2.14583839e+01,  1.83349520e+01],
      [ 1.88737767e+00,  1.68049393e+01,  2.20897506e+01, ...,
        1.43357519e+01,  2.13314634e+01,  2.21233522e+01],
      ...,
      [ 2.49704127e+03,  1.16100973e+01,  1.04631943e+01, ...,
        2.11507602e+01,  1.71370365e+01,  1.61821579e+01],
      [ 2.49792535e+03,  1.71125606e+01,  1.99157080e+01, ...,
        2.03126143e+01,  1.93389599e+01,  2.00825014e+01],
      [ 2.49889826e+03,  2.93423978e+01,  1.81761100e+01, ...,
        2.77286314e+01,  3.41596238e+01,  2.63240129e+01]])}
{5: array([[-1.07620749e-01,  2.24447056e+01,  2.10528209e+01, ...,
        1.90015237e+01,  2.23188490e+01,  1.96076932e+01],
      [ 9.28684513e-01,  2.32577490e+01,  2.09330989e+01, ...,
        1.90570959e+01,  2.14583839e+01,  1.83349520e+01],
      [ 1.88737767e+00,  1.68049393e+01,  2.20897506e+01, ...,
        1.43357519e+01,  2.13314634e+01,  2.21233522e+01],
      ...,
      [ 2.49704127e+03,  1.16100973e+01,  1.04631943e+01, ...,
        2.11507602e+01,  1.71370365e+01,  1.61821579e+01],
      [ 2.49792535e+03,  1.71125606e+01,  1.99157080e+01, ...,
        2.03126143e+01,  1.93389599e+01,  2.00825014e+01],
      [ 2.49889826e+03,  2.93423978e+01,  1.81761100e+01, ...,
        2.77286314e+01,  3.41596238e+01,  2.63240129e+01]]), 10: array([[-9.95872382e-02,  3.25100519e+01,  1.67802355e+01, ...,
        2.31607889e+01,  3.36130073e+01,  1.99024828e+01],
      [ 9.58037070e-01,  2.55662963e+01,  1.96008230e+01, ...,
        1.70011477e+01,  2.29077375e+01,  2.04950582e+01],
      [ 1.89920732e+00,  1.57489494e+01,  3.11341115e+01, ...,
        2.38163252e+01,  1.46273763e+01,  2.50563318e+01],
      ...,
      [ 2.49703548e+03,  7.92965753e+00,  1.68138944e+01, ...,
        2.10333162e+01,  8.59394838e+00,  1.40109375e+01],
      [ 2.49798110e+03,  2.71341779e+01,  1.33807276e+01, ...,
        3.49504840e+01,  3.43466612e+01,  3.29579402e+01],
      [ 2.49886897e+03,  4.37464312e+01, -5.89033506e+00, ...,
        1.84930292e+01,  6.81980904e+01,  2.44978595e+01]])}
```

Exploratory data analysis on SVD data with 5 columns.

- Shape of data: (2500, 51)
- There are no null values in the dataset.
- info() method prints information and description of the DataFrame.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 51 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Unnamed: 0   2500 non-null   int64
 1   1            2500 non-null   float64
 2   2            2500 non-null   float64
 3   3            2500 non-null   float64
 4   4            2500 non-null   float64
 5   5            2500 non-null   float64
 6   6            2500 non-null   float64
 7   7            2500 non-null   float64
 8   8            2500 non-null   float64
 9   9            2500 non-null   float64
 10  10           2500 non-null   float64
 11  11           2500 non-null   float64
 12  12           2500 non-null   float64
 13  13           2500 non-null   float64
 14  14           2500 non-null   float64
 15  15           2500 non-null   float64
 16  16           2500 non-null   float64
 17  17           2500 non-null   float64
 18  18           2500 non-null   float64
 19  19           2500 non-null   float64
 20  20           2500 non-null   float64
 21  21           2500 non-null   float64
 22  22           2500 non-null   float64
 23  23           2500 non-null   float64
 24  24           2500 non-null   float64
 25  25           2500 non-null   float64
 26  26           2500 non-null   float64
 27  27           2500 non-null   float64
 28  28           2500 non-null   float64
 29  29           2500 non-null   float64
 30  30           2500 non-null   float64
 31  31           2500 non-null   float64
 32  32           2500 non-null   float64
 33  33           2500 non-null   float64
 34  34           2500 non-null   float64
 35  35           2500 non-null   float64
 36  36           2500 non-null   float64
 37  37           2500 non-null   float64
 38  38           2500 non-null   float64
 39  39           2500 non-null   float64
 40  40           2500 non-null   float64
 41  41           2500 non-null   float64
 42  42           2500 non-null   float64
 43  43           2500 non-null   float64
 44  44           2500 non-null   float64
 45  45           2500 non-null   float64
 46  46           2500 non-null   float64
 47  47           2500 non-null   float64
 48  48           2500 non-null   float64
 49  49           2500 non-null   float64
 50  50           2500 non-null   float64
dtypes: float64(50), int64(1)
memory usage: 996.2 KB
```

1. Loading and Understanding the SVD Data:

- Given the SVD-reduced dataset with 5 columns, let's start by loading and getting an initial sense of the data's structure, distribution, and summary statistics.

2. Summary Statistics:

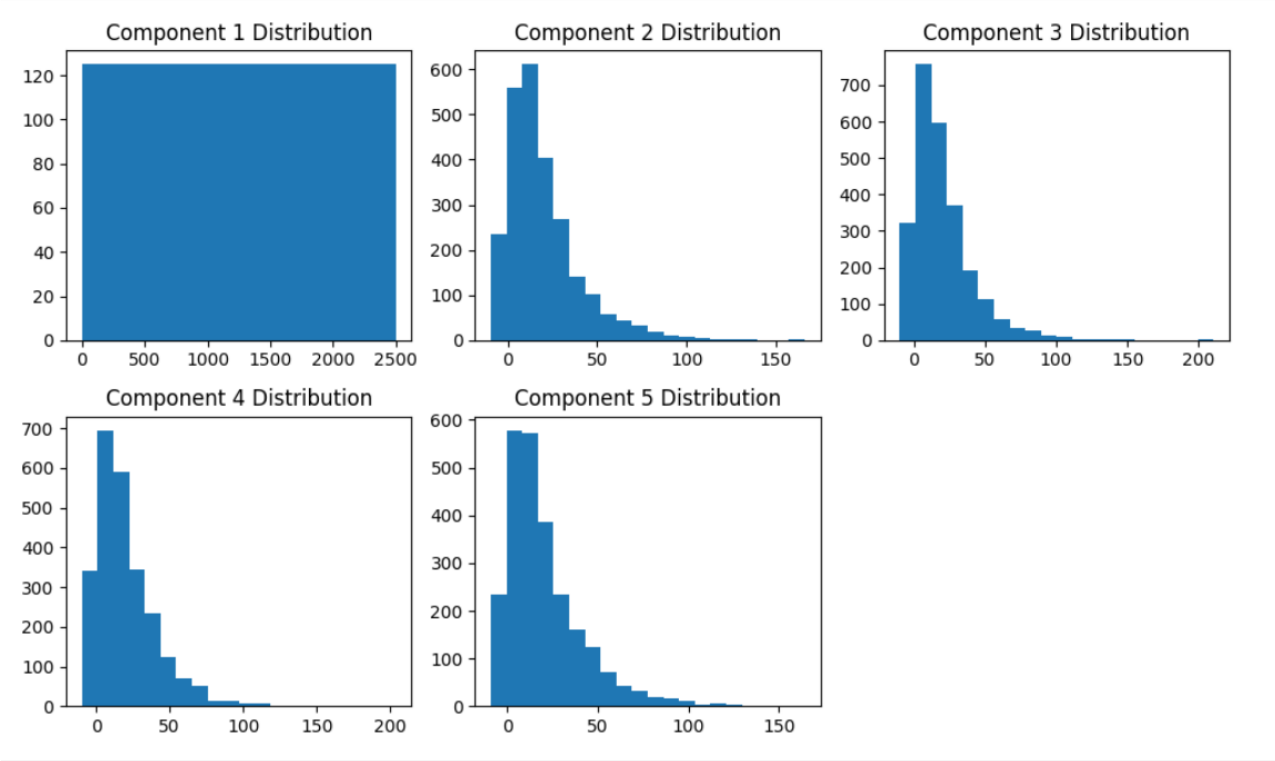
Summary statistics provide a concise overview of key characteristics within a dataset, such as measures of central tendency (mean, median) and dispersion (standard deviation, range), offering insights into its distribution and variability.

```
Results for SVD with 5 columns:
Shape of SVD result: (2500, 5)

Summary Statistics:
       Component_0  Component_1  Component_2  Component_3  Component_4
count  2500.000000  2500.000000  2500.000000  2500.000000  2500.000000
mean   1249.500582    19.376107    20.018453    20.289183    20.161816
std     721.831147     5.778854     6.498541     6.802788     9.085621
min      -0.107621     2.895804    -5.641592    -0.271409   -25.351322
25%     624.780222    15.390903    15.518247    15.641033    14.245183
50%    1249.508600    19.049057    19.619961    19.913724    19.870859
75%    1874.147845    22.795711    24.087598    24.310945    25.658667
max    2498.898256    49.885642    46.509324    52.386045    61.422459
```

3. Exploring Data Distribution and Relationships:

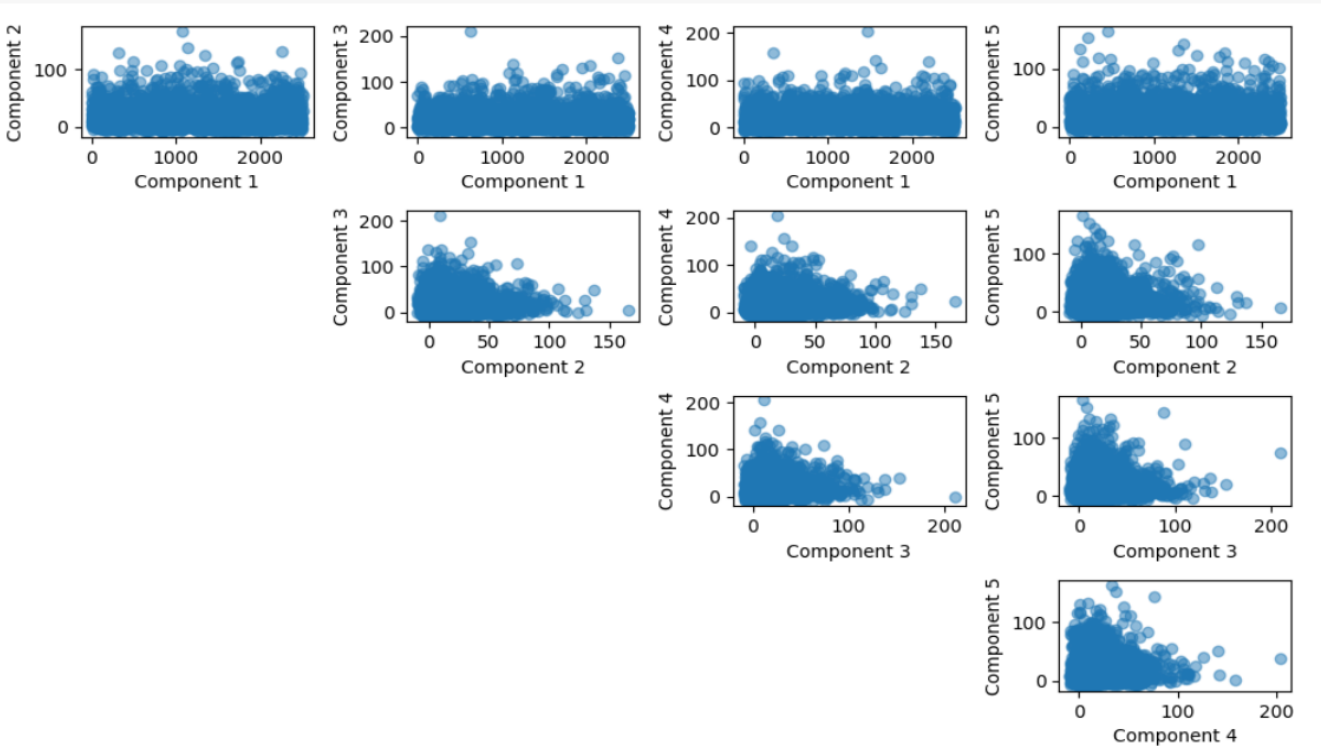
a. Distribution of Components:

- Visualize the distribution of each component using histograms or density plots to understand their spread and identify potential outliers.

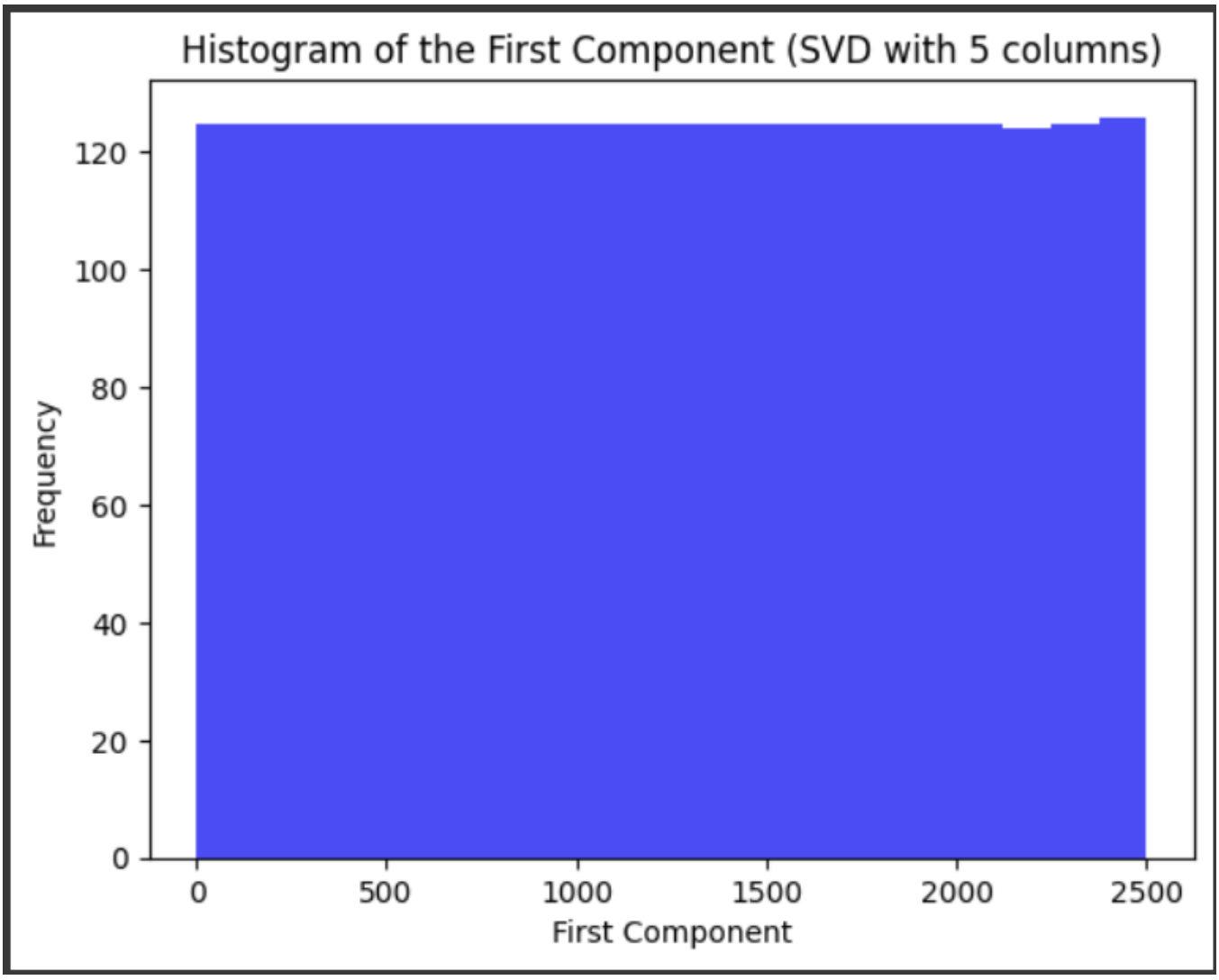


b. Pairwise Relationships:

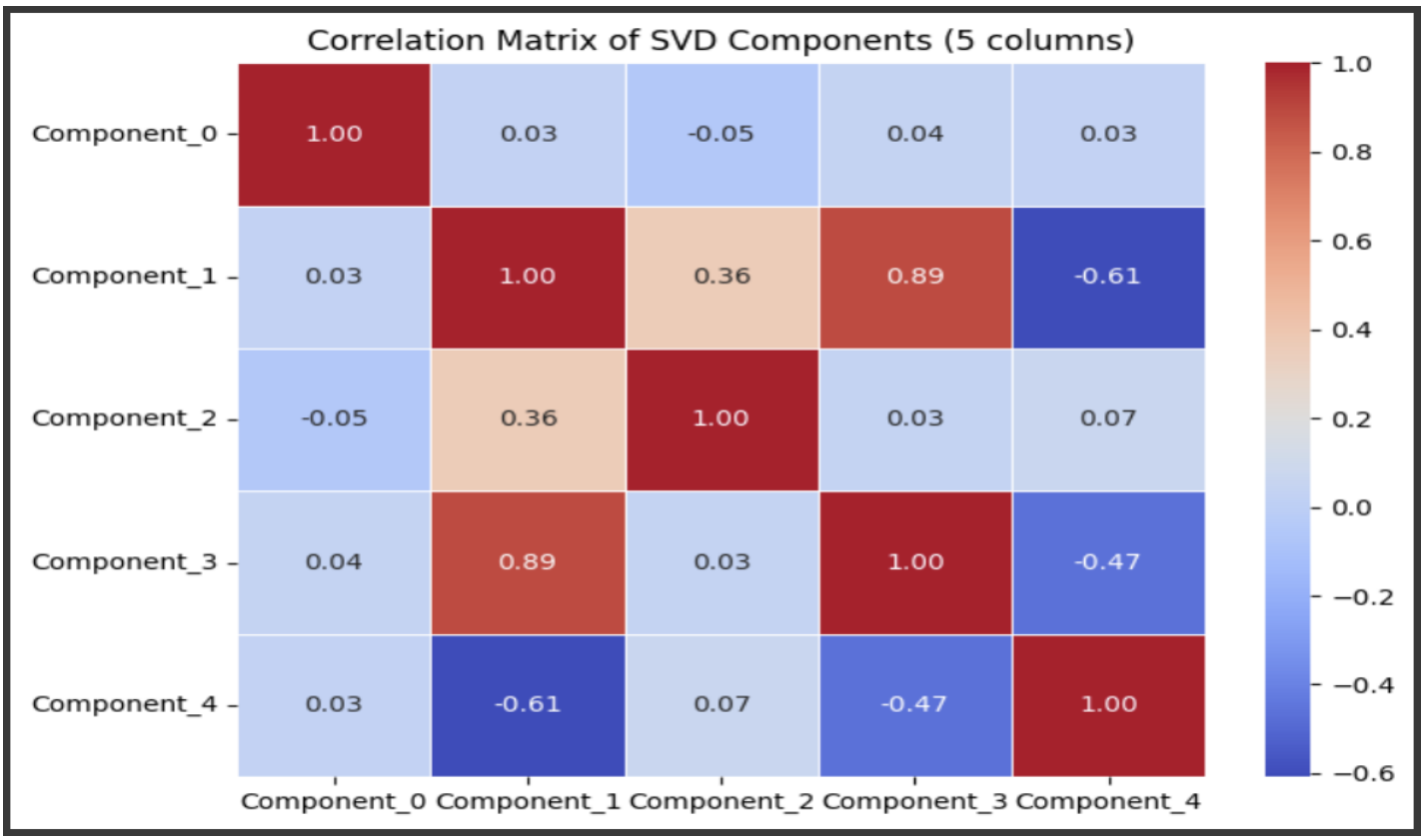
- Explore pairwise relationships between components using scatter plots to identify any linear relationships or clusters within the reduced space.



4. Histogram of first Component (SVD with 5 columns)



5. Correlation Matrix of SVD Components (5 columns)



Conclusion:

PCA and SVD are closely related, with PCA being a specific application of SVD. PCA focuses on finding orthogonal components that capture maximum variance, while SVD is a more general matrix factorization technique useful for various applications but computationally more intensive. The choice between them depends on the specific task's requirements, with PCA preferred for simplicity and speed, and SVD for its versatility in certain scenarios.

Problem statement - 3:

Use linear regression and LDA on data 3 to predict whether a given suburb has a crime rate above or below the median. Compare the findings from different methods.

Data:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

The Boston data frame has 506 rows and 14 columns. This data frame contains the following columns:

1) **crim**: Per capita crime rate by town.
2) **zn**: Proportion of residential land zoned for lots over 25,000 sq.ft.
3) **indus**: Proportion of non-retail business acres per town.
4) **chas**: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise).
5) **nox**: Nitrogen oxides concentration (parts per 10 million).
6) **rm**: Average number of rooms per dwelling.
7) **age**: proportion of owner-occupied units built prior to 1940.
8) **dis**: Weighted mean of distances to five Boston employment centers.
9) **rad**: Index of accessibility to radial highways.
10) **tax**: Full-value property-tax rate per $10,000.

11)**ptratio**: Pupil-teacher ratio by town.

12)**black**: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town.

13)**lstat**: Lower status of the population (percent).

14)**medv**: Median value of owner-occupied homes in $1000s.

Exploratory Data Analysis (EDA):

- Shape of data: (506 x 14)
- There are no null values in the dataset.
- info() method prints information and description of the DataFrame.

```
Information about the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

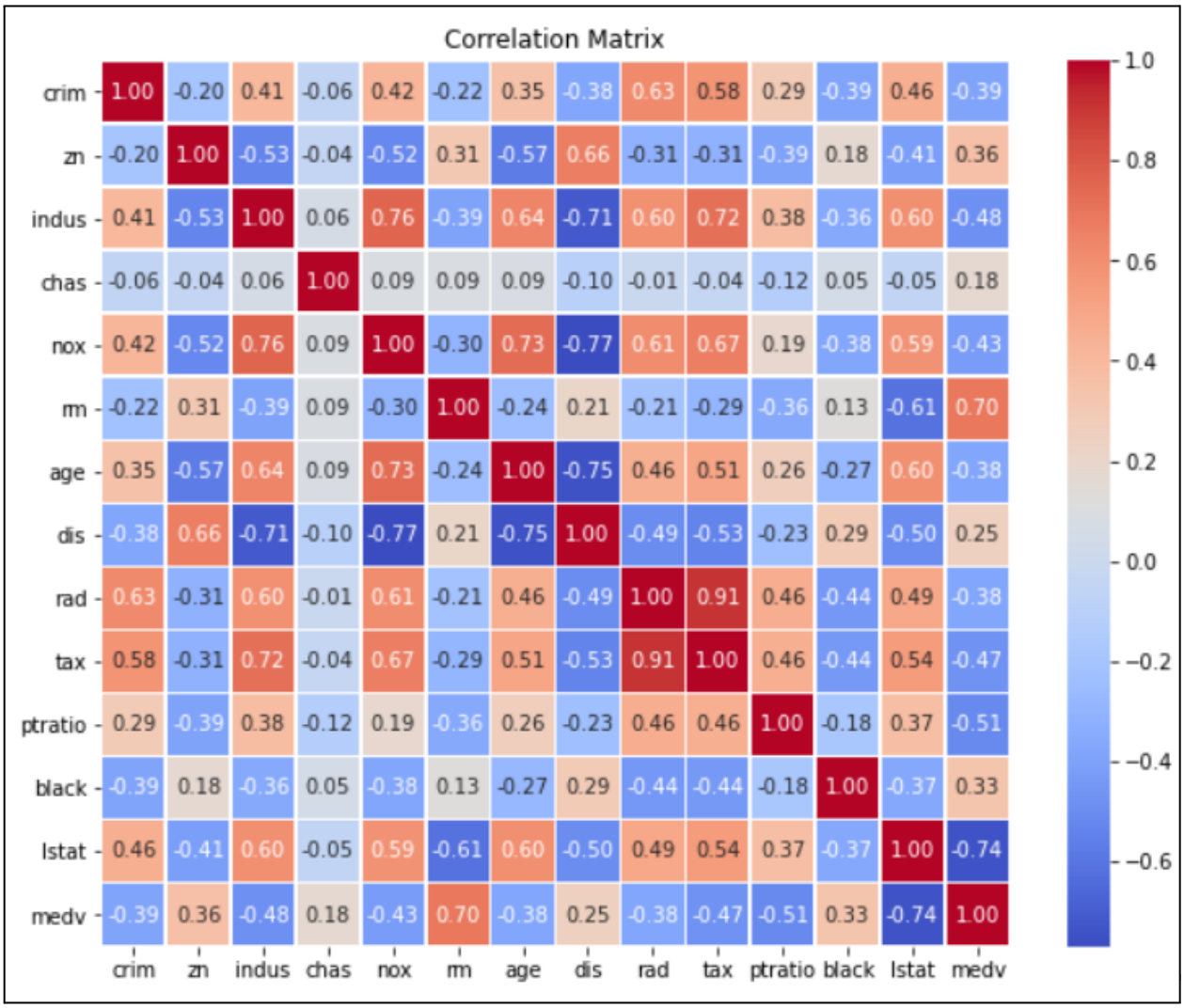
- describe() method returns the summary of the data in the DataFrame. It describes the following numerical data columns, i.e:
    - ❖ count - The number of not-empty values.
    - ❖ mean - The average (mean) value.
    - ❖ std - The standard deviation.
    - ❖ min - the minimum value.
    - ❖ 25% - The 25% percentile.
    - ❖ 50% - The 50% percentile.
    - ❖ 75% - The 75% percentile
    - ❖ max - the maximum value.

```
Summary of numerical columns:
              crim          zn       indus        chas         nox          rm  \
count   506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean      3.613524   11.363636   11.136779    0.069170    0.554695    6.284634
std       8.601545   23.322453    6.860353    0.253994    0.115878    0.702617
min       0.006320    0.000000    0.460000    0.000000    0.385000    3.561000
25%       0.082045    0.000000    5.190000    0.000000    0.449000    5.885500
50%       0.256510    0.000000    9.690000    0.000000    0.538000    6.208500
75%       3.677083   12.500000   18.100000    0.000000    0.624000    6.623500
max      88.976200  100.000000   27.740000    1.000000    0.871000    8.780000

               age         dis         rad         tax     ptratio       black  \
count   506.000000  506.000000  506.000000  506.000000  506.000000  506.000000
mean     68.574901    3.795043    9.549407  408.237154   18.455534  356.674032
std      28.148861    2.105710    8.707259  168.537116    2.164946   91.294864
min       2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%      45.025000    2.100175    4.000000  279.000000   17.400000  375.377500
50%      77.500000    3.207450    5.000000  330.000000   19.050000  391.440000
75%      94.075000    5.188425   24.000000  666.000000   20.200000  396.225000
max     100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            lstat        medv
count  506.000000  506.000000
mean    12.653063   22.532806
std      7.141062    9.197104
min      1.730000    5.000000
25%      6.950000   17.025000
50%     11.360000   21.200000
75%     16.955000   25.000000
max     37.970000   50.000000
```

Correlation Matrix:

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
crim	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	-0.385064	0.455621	-0.388305
zn	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.175520	-0.412995	0.360445
indus	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	-0.356977	0.603800	-0.483725
chas	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.048788	-0.053929	0.175260
nox	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	-0.380051	0.590879	-0.427321
rm	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847	-0.292048	-0.355501	0.128069	-0.613808	0.695360
age	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022	0.506456	0.261515	-0.273534	0.602339	-0.376955
dis	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588	-0.534432	-0.232471	0.291512	-0.496996	0.249929
rad	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000	0.910228	0.464741	-0.444413	0.488676	-0.381626
tax	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	-0.441808	0.543993	-0.468536
ptratio	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	-0.177383	0.374044	-0.507787
black	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.000000	-0.366087	0.333461
lstat	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	-0.366087	1.000000	-0.737663
medv	-0.388305	0.360445	-0.483725	0.175260	-0.427321	0.695360	-0.376955	0.249929	-0.381626	-0.468536	-0.507787	0.333461	-0.737663	1.000000

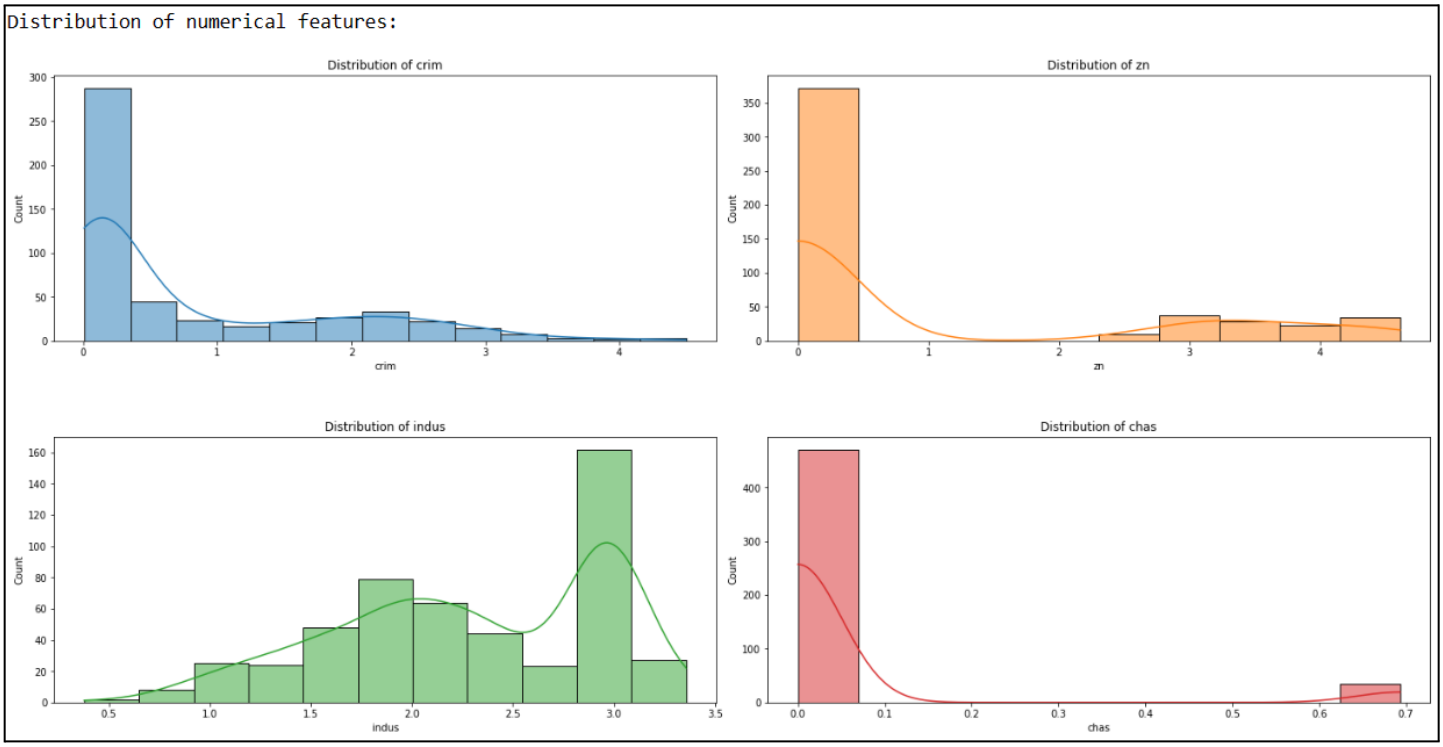
Using Heat Map

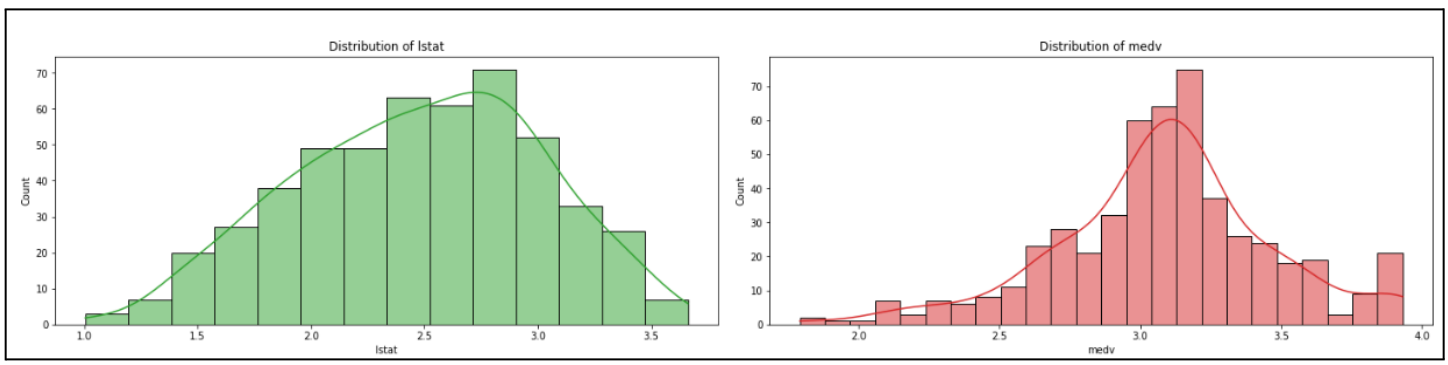


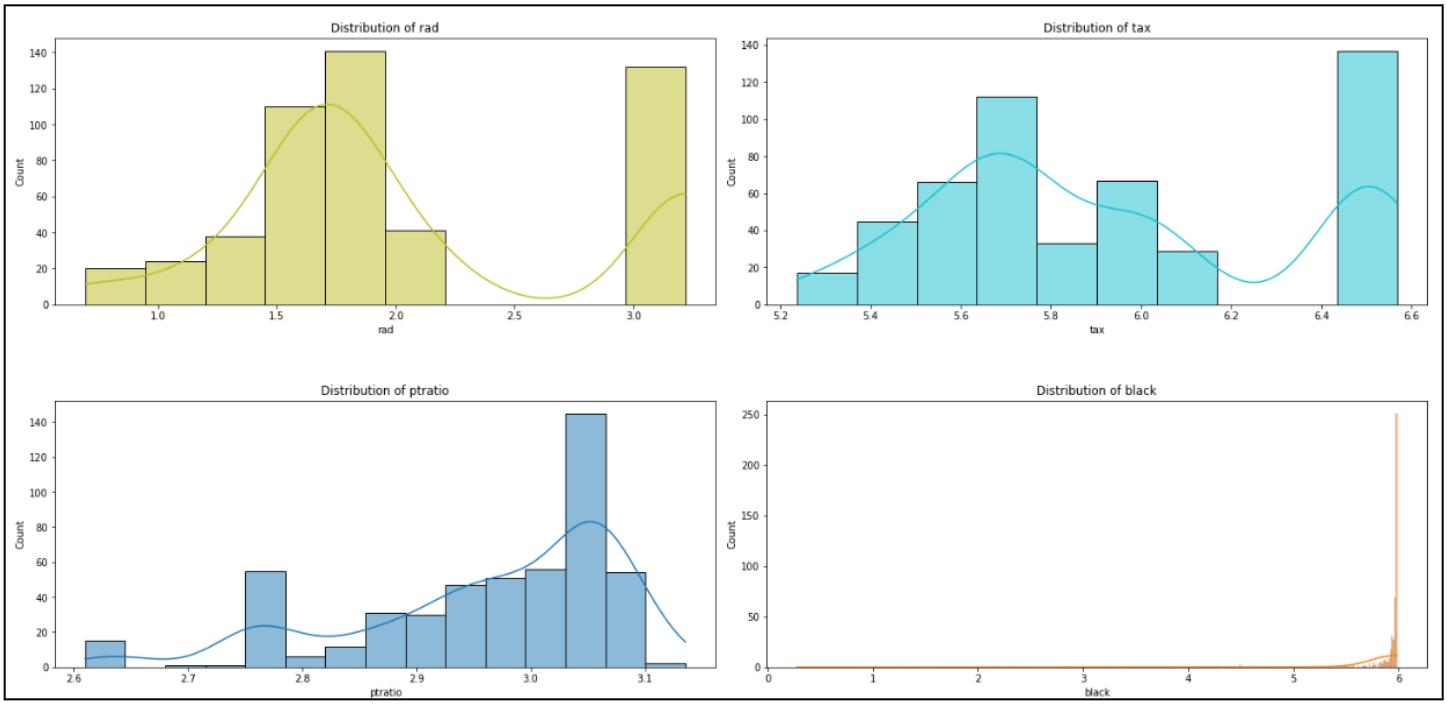
Correlation matrix and heatmap were made and it can be observed that most of the parameters are not well correlated. Only 2 of the parameters have correlation value above 0.75 (nox and indus, correlation value=0.76), so we concluded that all the parameters could be kept and feature scaling is not required.

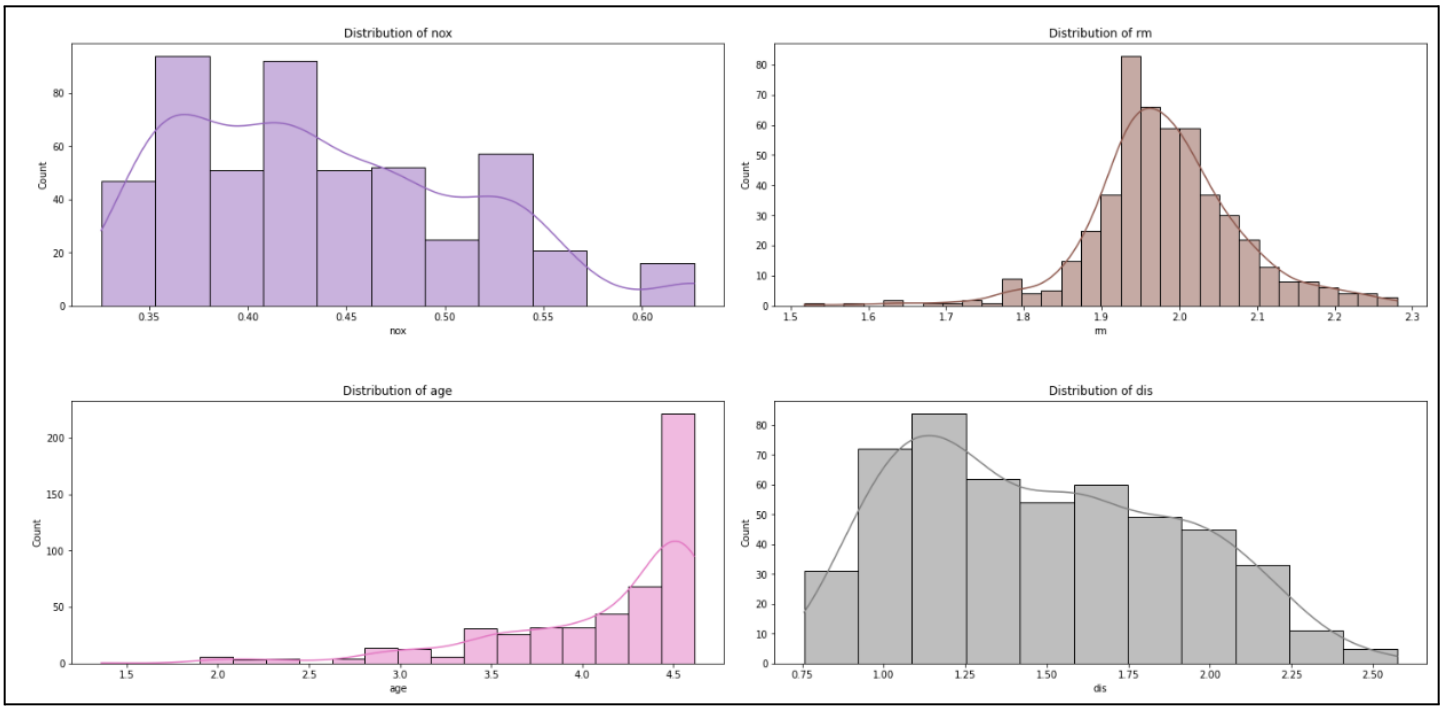
Histograms to show the distribution of features:

The histogram is a popular graphing tool. It is used to summarize discrete or continuous data that are measured on an interval scale. It shows the distribution of data columns.





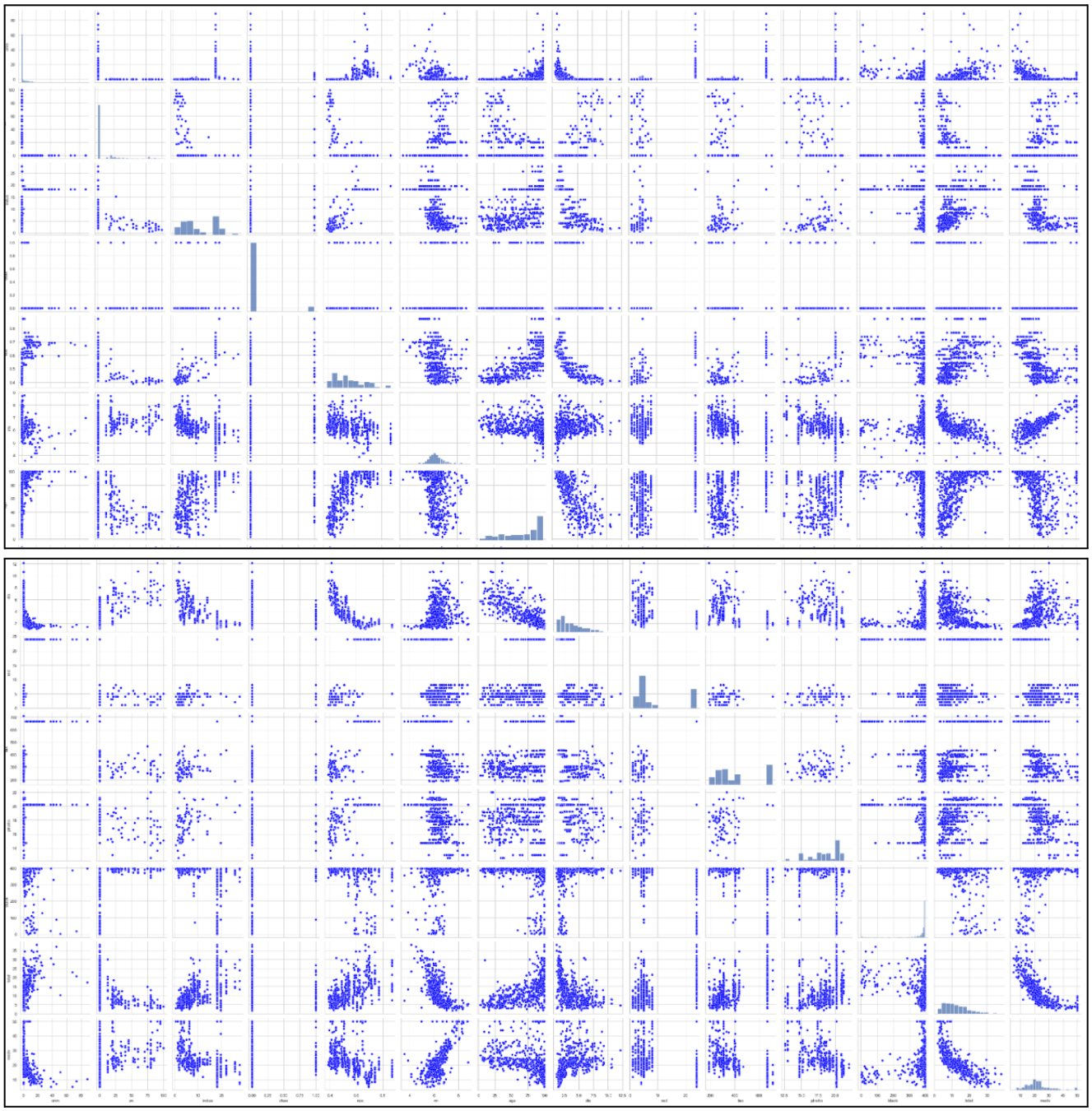




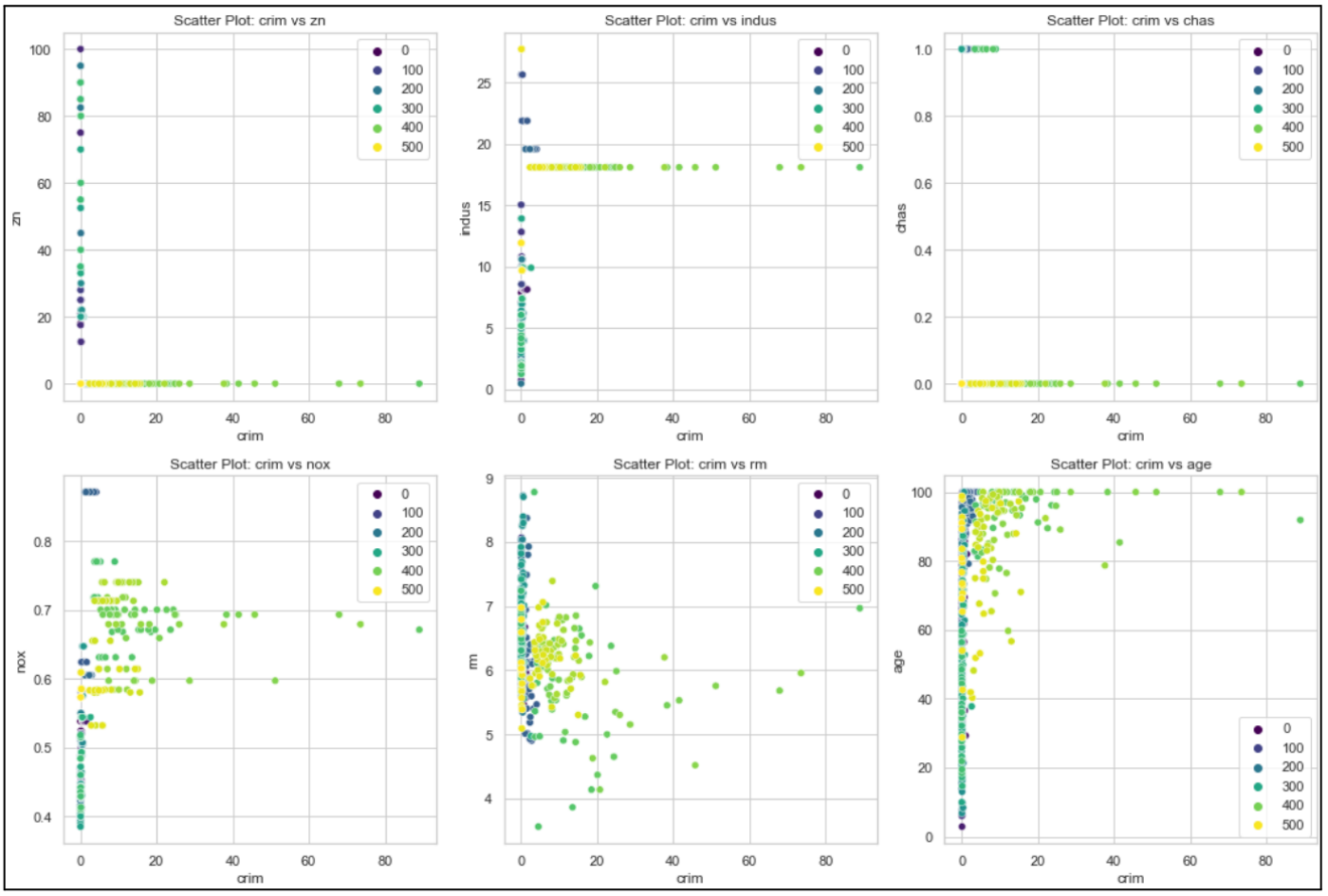
From the histogram plots, it could be observed clearly that 3 datasets are left skewed, 2 right skewed and 3 normally distributed.

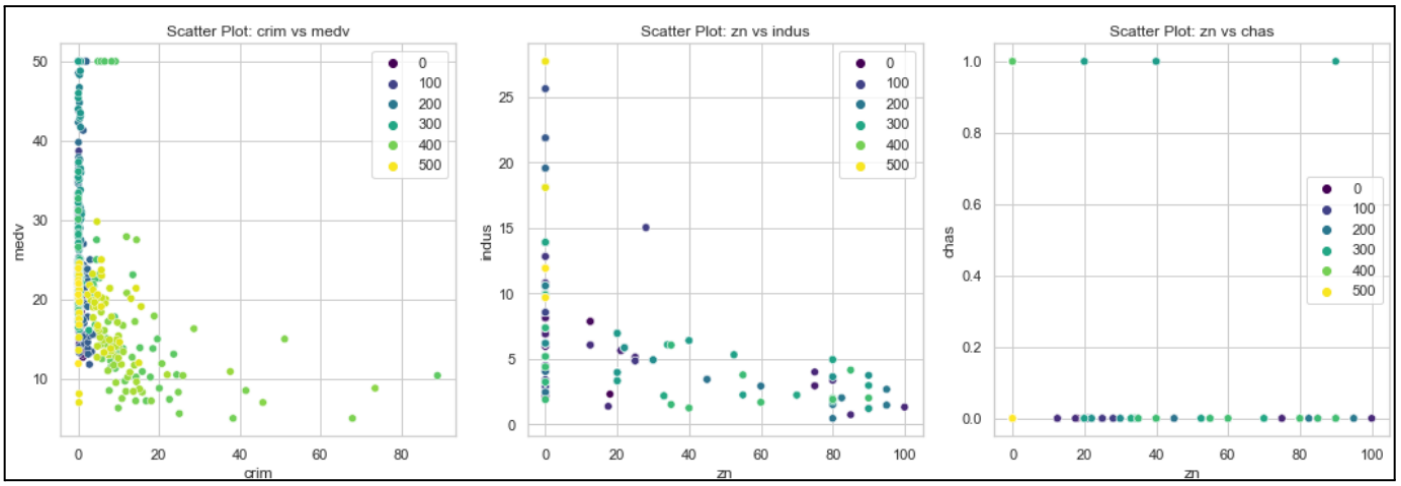
Pair Plots for all features:

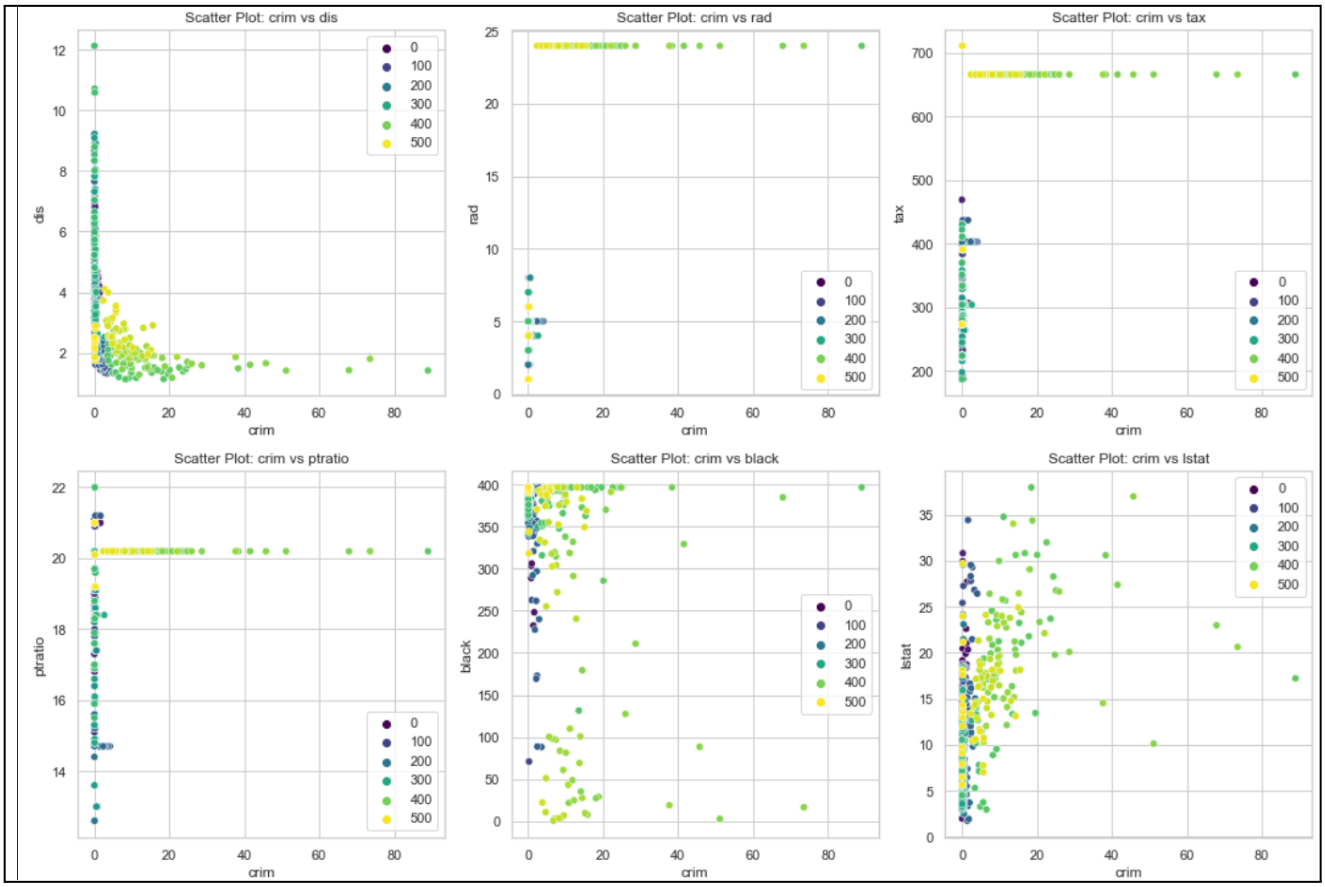
It provides a quick and comprehensive view of how different features are distributed and how they relate to each other.



Scatter-Plots:







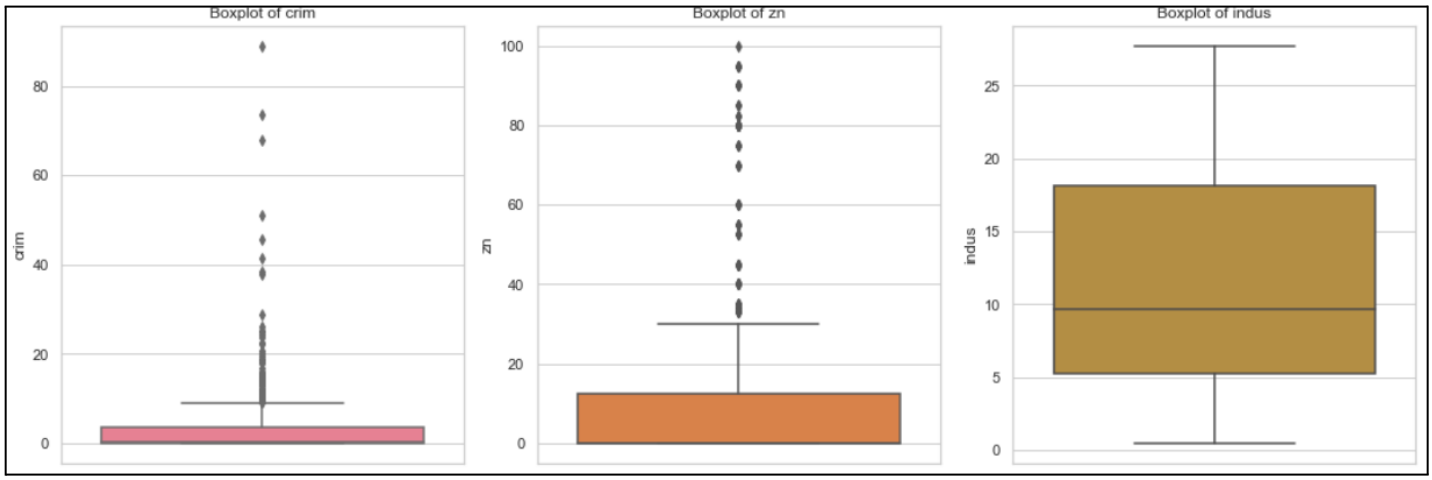
Outlier Detection:

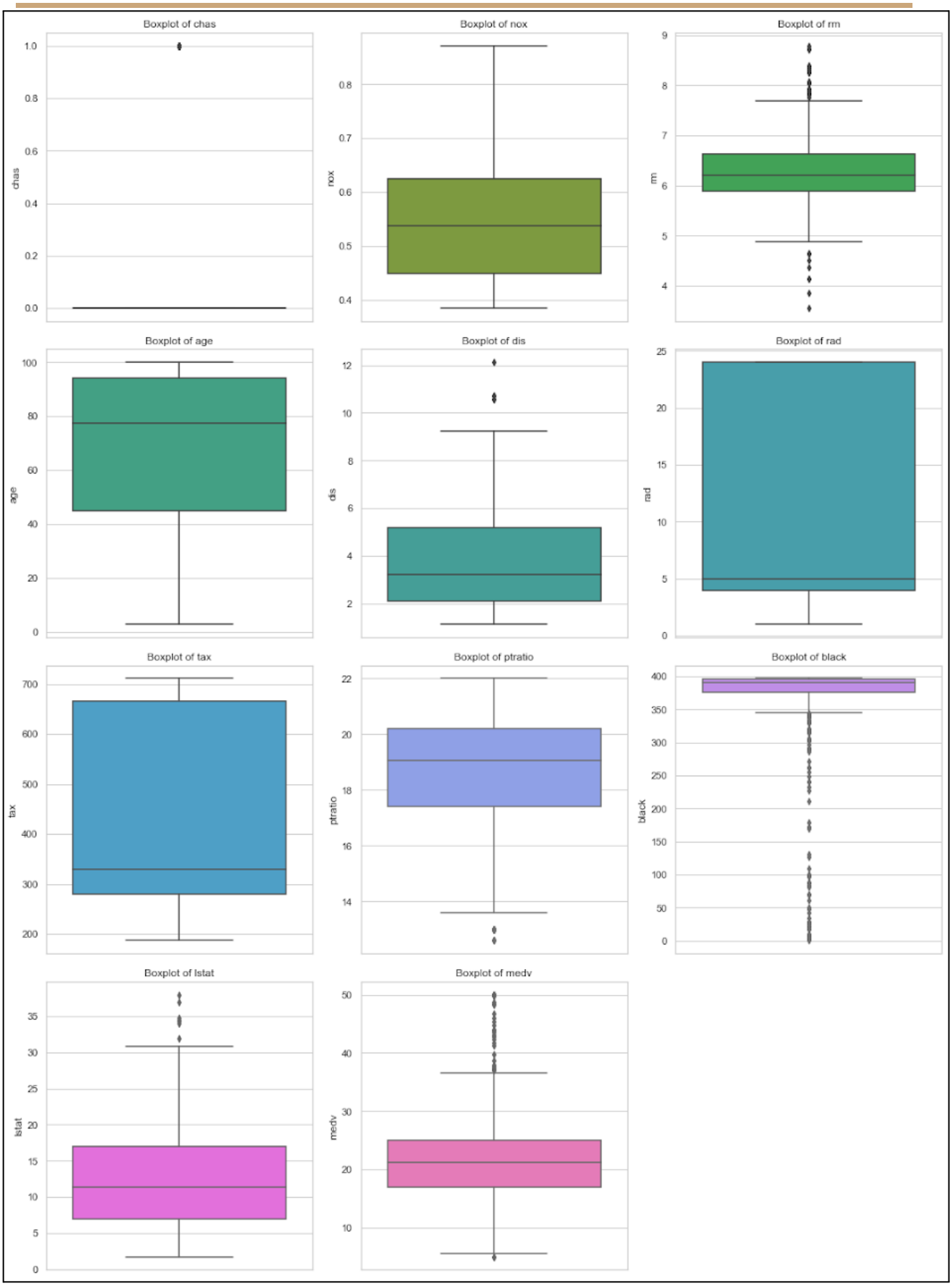
```
Detected Outliers:
        crim   zn  indus  chas    nox     rm    age     dis  rad  tax  \
367  13.52220  0.0  18.10     0  0.631  3.863  100.0  1.5106   24  666
371   9.23230  0.0  18.10     0  0.631  6.216  100.0  1.1691   24  666
373  11.10810  0.0  18.10     0  0.668  4.906  100.0  1.1742   24  666
374  18.49820  0.0  18.10     0  0.668  4.138  100.0  1.1370   24  666
375  19.60910  0.0  18.10     0  0.671  7.313   97.9  1.3163   24  666
..        ...  ...    ...   ...    ...    ...    ...     ...  ...  ...
161   1.46336  0.0  19.58     0  0.605  7.489   90.8  1.9709    5  403
179   0.05780  0.0   2.46     0  0.488  6.980   58.4  2.8290    3  193
182   0.09103  0.0   2.46     0  0.488  7.155   92.2  2.7006    3  193
228   0.29819  0.0   6.20     0  0.504  7.686   17.0  3.3751    8  307
368   4.89822  0.0  18.10     0  0.631  4.970  100.0  1.3325   24  666
```

	ptratio	black	lstat	medv
367	20.2	131.42	13.33	23.1
371	20.2	366.15	9.53	50.0
373	20.2	396.90	34.77	13.8
374	20.2	396.90	37.97	13.8
375	20.2	396.90	13.44	15.0
..	...	...	...	...
161	14.7	374.43	1.73	50.0
179	17.8	396.90	5.04	37.2
182	17.8	394.12	4.82	37.9
228	17.4	377.51	3.92	46.7
368	20.2	375.52	3.26	50.0

[238 rows x 14 columns]

To visualize the detected outliers, the following box plot is plotted.




Boxplot of chas
chas
Boxplot of nox
nox
Boxplot of rm
rm
Boxplot of age
age
Boxplot of dis
dis
Boxplot of rad
rad
Boxplot of tax
tax
Boxplot of ptratio
ptratio
Boxplot of black
black
Boxplot of lstat
lstat
Boxplot of medv
medv

Data-preparation

- We add a new column ‘target’ which will be equal to ‘y’.
- We have to check only for crime rate so we define y as:
  y = True (if data[crim]>median(data[crim])
  y = False (if data[crim]<median(data[crim])
- After this, the dataset is split into 75:25 ratios for training and testing respectively.

Result Analysis of Model and Interpretation:

Metrics of Linear regression:

Coefficients: [0.01166075 -0.03559289 0.02818056 -0.0012968
0.22635726 0.0015621 0.07187933 0.02420881 0.14126721 -0.0225955
0.02705624 -0.02081112 0.02390561 0.09615458]

Intercept: 0.5092348284960418

Metrics for Training Data:
Mean Absolute Error: 0.14511873350923482
Mean Square Error: 0.14511873350923482
Root Mean Square Error: 0.380944528126123 4

Metrics for Test Data:
Mean Absolute Error: 0.11023622047244094
Mean Square Error: 0.11023622047244094
Root Mean Square Error: 0.332018403815874 3

Such low values of these errors show that the model is well fitted on the dataset and also, predicts the output well for new entries.

Accuracy Score

Accuracy Score	Linear Regression	LDA
Training Dataset	0.8548812664907651	0.8443271767810027
Testing Dataset	0.889763779527559	0.9133858267716536

We see that both the models have very high accuracy scores of 89% and 91% respectively on the training dataset. From the above accuracy scores, we see that LDA performs better compared to Linear Regression on the Testing dataset whereas on the training dataset, they are almost the same, but Linear Regression is a little better.

Positive and Negative Rates

Training Set

Training Dataset	Linear Regression	LDA
False Positive Rate	0.053763440860215055	0.06878306878306878
True Positive Rate	0.7668393782383419	0.7578947368421053
False Negative Rate	0.23316062176165803	0.24210526315789474
True Negative Rate	0.946236559139785	0.9312169312169312

Testing Set

Testing Dataset	Linear Regression	LDA
False Positive Rate	0.14925373134328357	0.015625
True Positive Rate	0.9333333333333333	0.8412698412698413
False Negative Rate	0.06666666666666667	0.15873015873015872
True Negative Rate	0.8507462686567164	0.9133858267716536

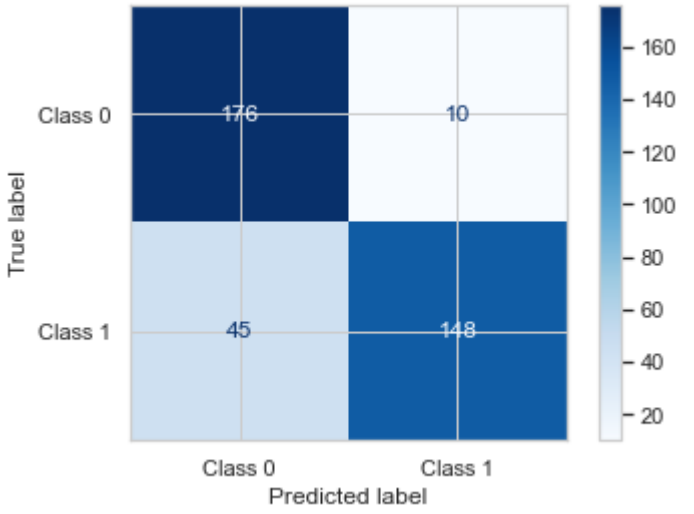
There is a trade-off between positive and negative rates. For some cases, Linear regression is better and for some, LDA works well.

Confusion Matrix

Linear Regression On Training Data:

[[176 10]

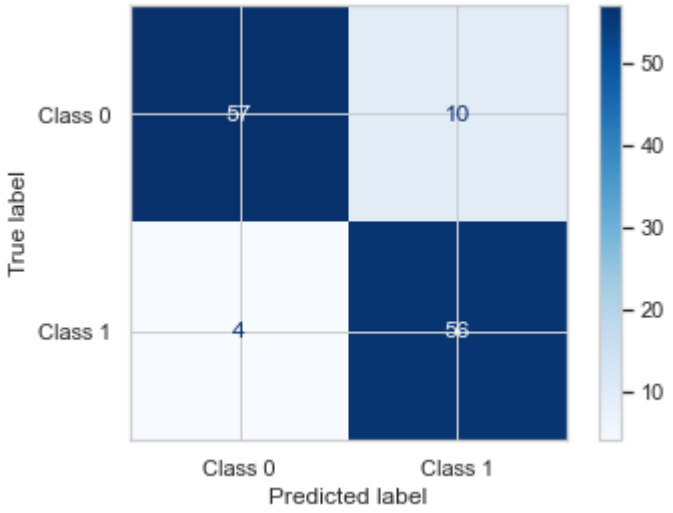
 [45 148]]



Linear Regression On Testing Data:

[[57 10]

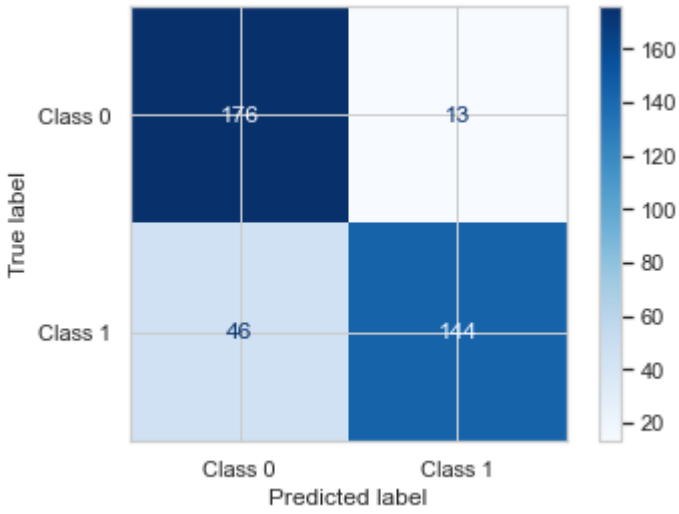
 [4 56]]



LDA On Training Data:

[[176 13]

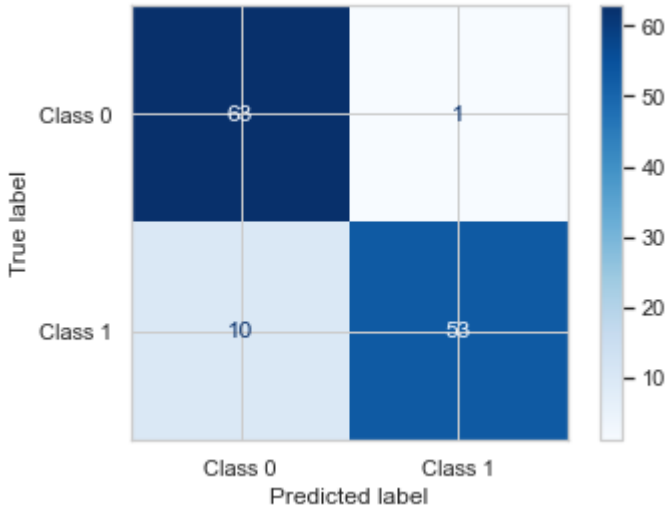
 [46 144]]



LDA On Testing Data:

[[63 1]

 [10 53]]



K-fold Cross-Validation Score:

Cross-Validation Score	Linear Regression	LDA
Original Dataset	0.5805933566884987	0.8418947777130654
Training Dataset	0.548640988387735	0.8285263157894736
Testing Dataset	0.4905147776794599	0.8735384615384616

Using this cross-validation score, we can clearly say that LDA is better than Linear Regression for this problem. There is a huge difference in the accuracy score and cross-validation score of Linear regression.

Conclusion:

- For the given question, we used two different models - LDA and Linear Regression- to train the dataset.
- We analyzed the model's performance using different metrics such as error values, confusion matrix, accuracy score, and cross-validation score.
- Based on these metrics, we can conclude that LDA is a better model than Linear regression for the given classification problem.
- We can also say that Linear regression performs far better for predicting YES and LDA works best for predicting NO.

Problem statement - 4:

Use the image given and compress it to 25%, 65% and 85% of the original image with SVD.

Image:

The image used is Image 2.jpg. The image shape is 1080 x 1920 x 3. Since it is an RGB image it has 3 matrices of 1080 x 1920, representing values for Red, Green and Blue.

Code Analysis:

- Each layer of Red, Green and Blue in the image is compressed separately.
- compress_rgb_image() function takes 2 parameters:
    - ❖ original_image: original_image which has to be compressed
    - ❖ compression_ratio: ratio of compressed image required to that of original image
- compress_gray_image_1() function takes 2 parameters:
    - ❖ original_image: original_image which has to be compressed
    - ❖ compression_ratio: ratio of compressed image required to that of original image
- compress_gray_image_2() function takes 2 parameters:
    - ❖ original_image: original_image which has to be compressed
    - ❖ compression_ratio: ratio of compressed image required to that of original image
- The following images are displayed in a grid of 2 x 2 for RGB compressed, Gray compressed by Method 1 and Method 2:
    - ❖ Original image
    - ❖ 25% compressed image
    - ❖ 65% compressed image
    - ❖ 85% compressed image
- The following images are displayed in a grid of 2 x 1 for RGB compressed to depict the difference after compressing:
    - ❖ Original image
    - ❖ 5% compressed image
- svd() function takes a matrix and returns the singular value decomposition of the matrix.

Exploratory Data Analysis (EDA):

- The shape of the image is 1080 x 1920 x 3.
    - ❖ RGB image with Red, Green and Blue layer.

```
original_image.shape
✓ 0.0s
(1080, 1920, 3)
```

- Image compressed to the following compression ratio:
    - ❖ 25% of the original image
    - ❖ 65% of the original image
    - ❖ 85% of the original image
- Length of the SVD columns to be included for the compression ratio using Method 1(Total Columns):
    - ❖ 5% -> 34
    - ❖ 25% -> 172
    - ❖ 65% -> 449
    - ❖ 85% -> 589

```
Optimal value of k for image to be compressed to 5.0 % using Method 1 is 34
```

```
Optimal value of k for image to be compressed to 25.0 % using Method 1 is 172
Optimal value of k for image to be compressed to 65.0 % using Method 1 is 449
Optimal value of k for image to be compressed to 85.0 % using Method 1 is 587
```

- Length of the SVD columns to be included for the compression ratio using Method 2(Total Columns):
    - ❖ 25% -> 1
    - ❖ 65% -> 45
    - ❖ 85% -> 297

```
Optimal value of k for image to be compressed to 25.0 % using Method 1 is 1
Optimal value of k for image to be compressed to 65.0 % using Method 1 is 45
Optimal value of k for image to be compressed to 85.0 % using Method 1 is 297
```

Methods for Finding k:

Method 1:

Considering the matrices that have to be stored for the compressed image. Well known standard method for image compression using SVD.

- Assuming original image has size of n x m for one layer i.e. either Red, Green or Blue, because the process is similar for all three.
- Now to compress the image we need to store 3 matrices of size m x k, k, n x k, taking the compression ratio to be c.
- We get the optimal value for k to be:
    - k = c * (n * m) / (n + 1 + m)

Method 2:

Considering the weight of the singular values in a descending order. (Method done in Lab session), but gives poor and unreliable results.

- Assuming original image has size of n x m for one layer i.e. either Red, Green or Blue, because the process is similar for all three.
- The singular values generated by the Singular Value Decomposition of original images is S (sorted in descending order), and compression ratio is c.
- We get the optimal value for k to be:
    - k = Sum(S[i to k])
        - where
            - Sum(S[i to k]) / Sum(S[i to n]) >= c

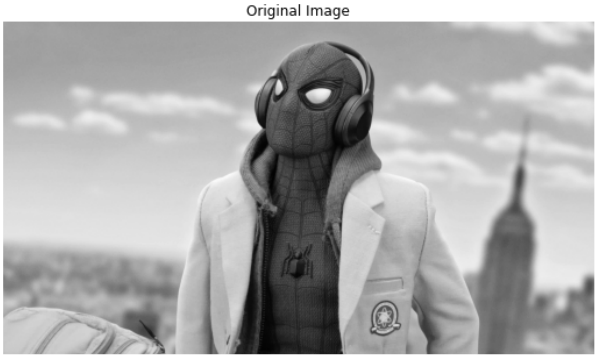
RGB Compressed Images using Method 1:

The above results are for images that are 25%, 65% and 85% of the original image.

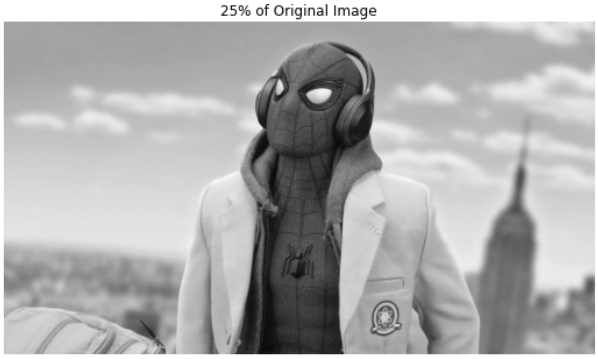
For 5% of the original image, the difference can be clearly seen:

Gray Compressed using Method 1:

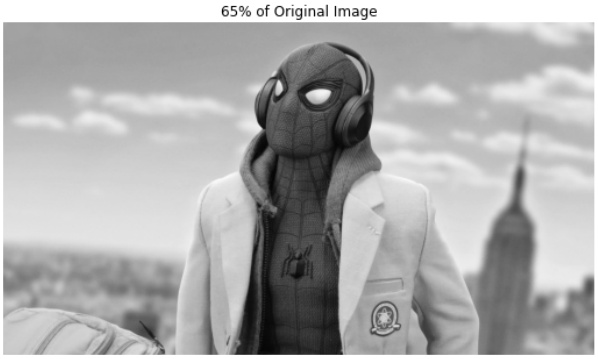
Original Image



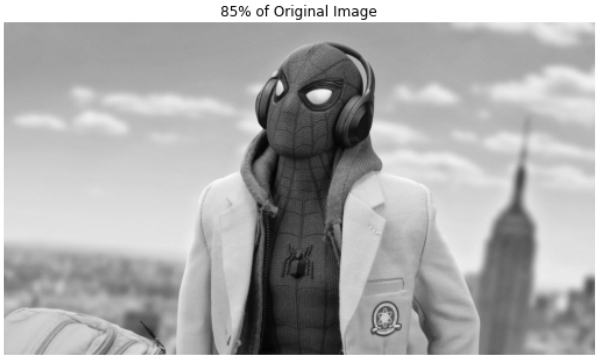
25% of Original Image



65% of Original Image



85% of Original Image

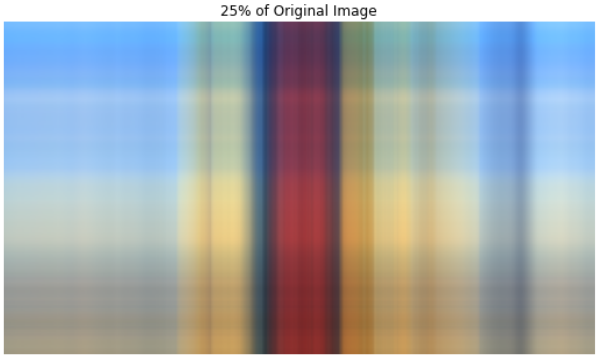


The results with Method 1 are very good and very little loss of information can be seen with this.

RGB Compressed Image using Method 2:

Original Image

25% of Original Image

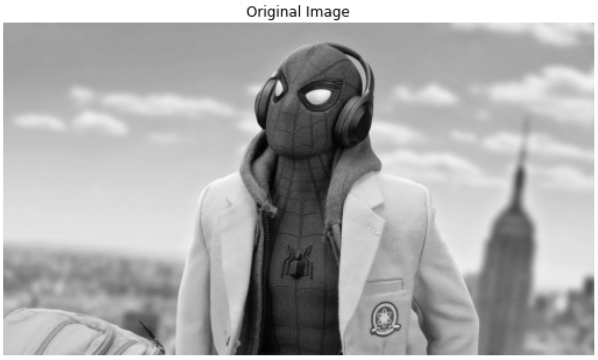


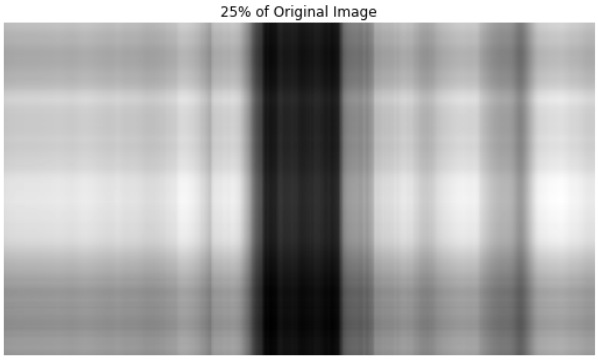
65% of Original Image

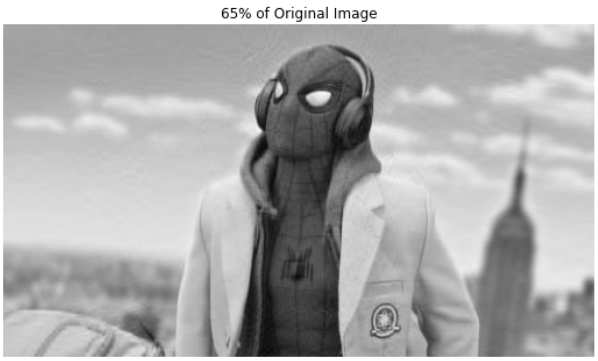
85% of Original Image

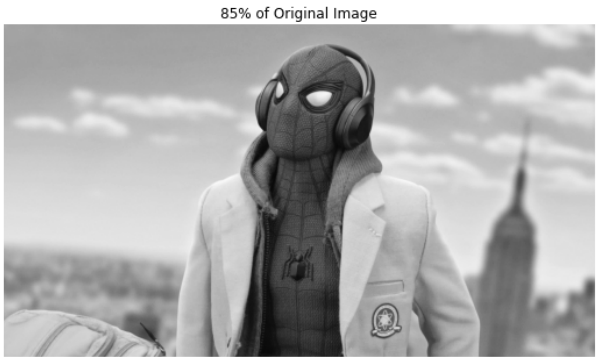
The results with Method 2 are very poor and the images are totally unpredictable with the singular values and their weights.

Gray Compressed using Method 2:









The results with Method 2 are very poor and a huge loss of information can be seen with this, in the 2nd case of 25% of the original image, where the image is distorted in a very unexpected way.

Conclusion

From the above analysis, we conclude that the images were compressed to 25%, 65% and 85% of the original image using SVD technique, and didn’t lose a lot of information with Method 1, but horrible results were seen with Method 2.

Therefore, we conclude that the Singular value Decomposition is a very powerful technique to compress images.