Degree project

Master's Programme in Information Technology – 120 credits

Project - Applied Linear Algebra

Course MA8017 - Engineering Mathematics, ht2, 2022

Applied linear algebra – 1.5hp

Halmstad - January, 2023
Kalhara Shabish Chithradewa

HALMSTAD UNIVERSITY

Table of Contents

List of Abbreviations

SVD	Singular Value Decomposition
PCA	Principal Component Analysis

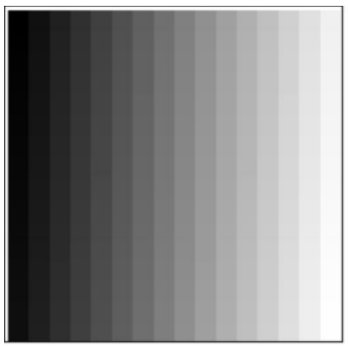
1 Introduction

This project covers basic linear algebra concepts and how it is applied to solve problems in two different scenarios. The primary focuses are image compression and the price prediction/estimation of houses using different linear algebra methods to follow all self-conducted problem analysis steps to prepare for future master research.

2 Problem 1 - Image compression

2.1 Problem statement

Images displayed on a computer screen are a collection of colour dots (pixels). This implies that any images are stored as a matrix. You can change that matrix to make the computer display a different picture: changing how dark or bright something is or flipping an image upside down. Each image matrix contains intensity coefficients for one colour. So, only one matrix is required for Black-and-White (grayscale) images (Figure 1). (Colour images have three matrices - one each of RGB (Red-Green-Blue) components.)



$$\begin{bmatrix}
0 & 16 & 32 & 48 & 64 & 80 & 96 & 112 & 128 & 144 & 160 & 176 & 192 & 208 & 224 & 240 \\
1 & 17 & 33 & 49 & 65 & 81 & 97 & 113 & 129 & 145 & 161 & 177 & 193 & 209 & 225 & 241 \\
2 & 18 & 34 & 50 & 66 & 82 & 98 & 114 & 130 & 146 & 162 & 178 & 194 & 210 & 226 & 242 \\
3 & 19 & 35 & 51 & 67 & 83 & 99 & 115 & 131 & 147 & 163 & 179 & 195 & 211 & 227 & 243 \\
4 & 20 & 36 & 52 & 68 & 84 & 100 & 116 & 132 & 148 & 164 & 180 & 196 & 212 & 228 & 244 \\
5 & 21 & 37 & 53 & 69 & 85 & 101 & 117 & 133 & 149 & 165 & 181 & 197 & 213 & 229 & 245 \\
6 & 22 & 38 & 54 & 70 & 86 & 102 & 118 & 134 & 150 & 166 & 182 & 198 & 214 & 230 & 246 \\
7 & 23 & 39 & 55 & 71 & 87 & 103 & 119 & 135 & 151 & 167 & 183 & 199 & 215 & 231 & 247 \\
8 & 24 & 40 & 56 & 72 & 88 & 104 & 120 & 136 & 152 & 168 & 184 & 200 & 216 & 232 & 248 \\
9 & 25 & 41 & 57 & 73 & 89 & 105 & 121 & 137 & 153 & 169 & 185 & 201 & 217 & 233 & 249 \\
10 & 26 & 42 & 58 & 74 & 90 & 106 & 122 & 138 & 154 & 170 & 186 & 202 & 218 & 234 & 250 \\
11 & 27 & 43 & 59 & 75 & 91 & 107 & 123 & 139 & 155 & 171 & 187 & 203 & 219 & 235 & 251 \\
12 & 28 & 44 & 60 & 76 & 92 & 108 & 124 & 140 & 156 & 172 & 188 & 204 & 220 & 236 & 252 \\
13 & 29 & 45 & 61 & 77 & 93 & 109 & 125 & 141 & 157 & 173 & 189 & 205 & 221 & 237 & 253 \\
14 & 30 & 46 & 62 & 78 & 94 & 110 & 126 & 142 & 158 & 174 & 190 & 206 & 222 & 238 & 254 \\
15 & 31 & 47 & 63 & 79 & 95 & 111 & 127 & 143 & 159 & 175 & 191 & 207 & 223 & 239 & 255
\end{bmatrix}$$

Figure 1: Gray scale presentation

Storing digital images requires large amounts of computer memory. Therefore, we always want to reduce memory storage without losing too much information from the image, i.e. trying to preserve quality. Image compression is also required to communicate/stream images faster. To do this, methods of linear algebra are used. There are several techniques to compress images, one of them is Principal Component Analysis (PCA) which can be performed using Singular Value Decomposition (SVD).

In this section, we will discuss how linear algebra can be used to compress images. We will discuss how SVD and PCA techniques are extensively used in the image compression process, saving the computer's memory. The basic idea here is that each image can be represented as a matrix, and we apply linear algebra (SVD and PCA) to get a reduced matrix out of this original matrix. The image corresponding to this reduced matrix requires much less storage space than the original image.

2.2 Background

2.2.1 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a matrix decomposition method that decomposes a matrix into a product of three matrices: a rectangular matrix, a diagonal matrix, and another rectangular matrix. It reduces a matrix to its simplest form, enabling us to identify patterns and trends in data and perform various matrix operations more efficiently.

The three matrices produced by SVD are:

U: a unitary matrix (meaning it satisfies $U * U' = U' * U = I$, where I is the identity matrix and $'$ denotes the transpose)

S: a diagonal matrix containing the singular values of the original matrix

V: an another unitary matrix

The original matrix X is a matrix of size m x n, can be reconstructed as:

$$X = U * S * V'$$

where $*$ denotes matrix multiplication and U is a matrix of size m x m, Sigma(S) is a diagonal matrix of size m x n, and V is a matrix of size n x n.

The SVD provides a numerically stable matrix decomposition that can use for various purposes. It provides stable compression and a practical demonstration of linear algebra as a compression tool. One of the immediate advantages of the SVD is that, unlike many matrix decompositions, it can be found for any matrix regardless of size or singularity. Furthermore, the SVD can be used to minimize the level of a matrix, which can be helpful for data compression.

In addition, another critical aspect of the SVD is highlighting the algorithm of principal component analysis (PCA), where high-dimensional data is decomposed into its most statistically defining factors. SVD and PCA have been applied to various problems in science and engineering.

Overall, the SVD is a powerful tool for analyzing and manipulating matrices. It has various applications in various fields, including data compression, machine learning, computer vision, and natural language processing.

2.2.2 Principal Component Analysis (PCA)

Karl Pearson invented Principal Component Analysis (PCA) in 1901, a dimensionality reduction technique used to project high-dimensional data onto a lower-dimensional space. It performs this by finding a new set of uncorrelated variables, called principal components, that capture the most variance in the data. In Addition, PCA can be seen as a statistical process for finding the best representation for a data set.

PCA first standardizes the data, so each variable contains a mean of zero and a standard deviation of one. It then calculates the covariance matrix, which measures the relationship between all pairs of variables. The eigenvectors of this covariance matrix represent the principal components, and the corresponding eigenvalues indicate the amount of variance explained by each component.

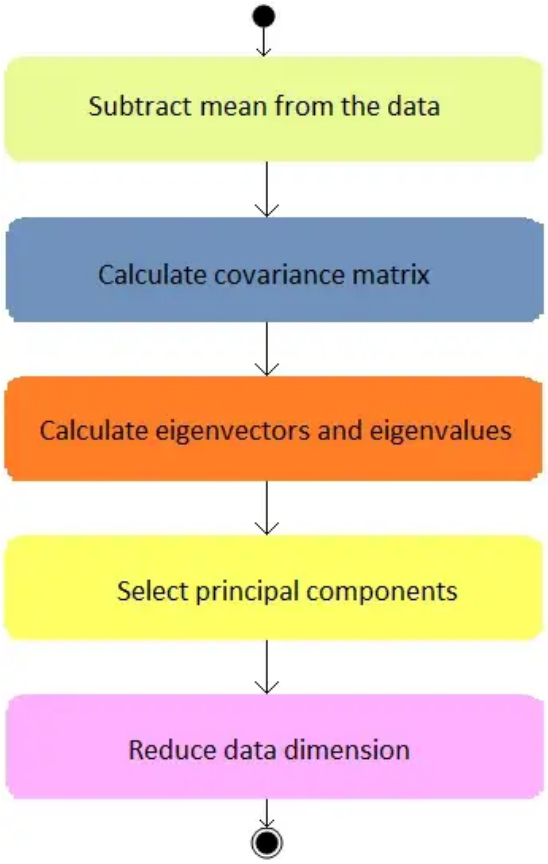


Figure 2: Process of PCA

As explained above, the process of PCA can be broken down into the following steps:

1. Standardize the data by subtracting the mean and scaling to unit variance. (Standardization)

$$X = \frac{x - mean(x)}{std(x)}$$

2. Calculate the covariance matrix (C) of the standardized data.

$$C = \frac{1}{(n-1)} * X' * X$$

where X is the original data matrix, X' is the transpose of X, and n is the number of observations in the data.

3. Calculate the eigenvectors and eigenvalues of the covariance matrix.

The eigenvectors and eigenvalues are calculated from the covariance matrix using the following equation:

$$C * V = \lambda * V$$

where C is the covariance matrix, V is the eigenvector, and λ is the eigenvalue.

4. Sort the eigenvalues by descending order and select the top k eigenvectors, where k is the number of dimensions you want to reduce the data to.

5. Use the selected eigenvectors to transform the original data into the new k-dimensional space. (Transformation)

The original data is transformed into the new k-dimensional space using the following equation:

$Xnew = X * W$

where Xnew is the transformed data, X is the original data, and W is the matrix of eigenvectors.

In order to decrease the dimensionality of the data, we can select the top k principal components, where k is the number of dimensions we want to retain. These principal components can then reconstruct the original data by multiplying them with the original standardized data.

PCA is a valuable data exploration, visualization, and feature selection tool. It can be used to identify patterns and relationships in the data and reduce the data's complexity without losing too much information. It is commonly used in data compression, machine learning, data mining, and statistical analysis.

2.2.3 Mathematical relationship between PCA and SVD

Principal Component Analysis and Singular Value Decomposition are closely related techniques used to reduce a data set's dimensionality. For instance, both can be applied to decompose any rectangular matrices. Both techniques find a set of linear combinations of the original variables that observe the most variance in the data. We can identify the relationship between both approaches by performing SVD on the covariance matrix on PCA.

PCA can be derived from SVD as follows:

Given a matrix X of size m x n, SVD decomposes it into the following three matrices:

$$X = U * S * V'$$

where U is a matrix of size m x m, S(sigma) is a diagonal matrix of size m x n, and V is a matrix of size n x n.

The covariant matrix C given as follows:

$$C = \frac{1}{(n-1)} * X' * X$$

where X is the original data matrix, X' is the transpose of X, and n is the number of observations in the data.

applying SVD to the data matrix X and X' as follows:

$$X = U * S * V'$$

$$X' = U' * S * V$$

and attempting to construct the covariance matrix from this decomposition gives

$$C = \frac{1}{(n-1)} * (U' * S * V) * (U * S * V')$$

U is a unitary matrix that satisfies $U * U' = U' * U = I$

$$C * V = \frac{1}{(n-1)} * S^2 * V$$

This is equivalent to PCA, where the matrix V is the matrix of eigenvectors, $\frac{1}{(n-1)} * S^2$ is the diagonal matrix of eigenvalues, and $C * V$ is the transformed data.

In summary, PCA and SVD are related to finding a set of linear combinations of the original variables that capture the most variance in the data. They both can be used for dimensionality reduction. However, SVD decomposes the data matrix into three separate matrices, while PCA directly finds the eigenvectors and eigenvalues of the covariance matrix.

2.3 Method

2.3.1 Data Collection

The provided dataset consists of 400 grayscale images of 40 people, and each person has ten different images of their faces with different impressions. Each image has a resolution of 112 x 92 pixels.

2.3.2 Image Compression

In this phase, perform PCA on the provided data set of the images to compress them by applying a few components associated with large eigenvalues and re-

constructing faces using only these few components as a compression technique.

The following Python codes perform the PCA to compress and reconstruct the data respectively.

The first step is importing the necessary libraries as Listing 1

```
import pandas as pd # for data processing, create data frames for the dataset (e.g. pd.DataFrame)
import numpy as np # for linear algebra
import matplotlib.pyplot as plt # for display the data set and the neccesary plots
import imageio.v2 as im # interface to read and write a wide range of image data
from glob import iglob  # locate the all of the image files that are present in the system.
from sklearn.decomposition import PCA # apply PCA to the dataset
```

Listing 1: Importing Python libraries

The next step is to read all the images and make a data frame (as a matrix) for all images, respectively, as in Figure 3 in Listing 2

```
faces_matrix = pd.DataFrame([]) # create a data frame to store the data of images
for path in iglob('att_faces/*/*.pgm'): # in this for loop iterates the folder that contains the images
    img= im.imread(path) # get the image file
    face = pd.Series(img.flatten(), name=path) # make a series(1 x 10304) of the image with the respective pixel value
    faces_matrix = faces_matrix.append(face, ignore_index=True) # append the image pixel values to data frame

faces_matrix # show the data frame that we are going to apply PCA
```

Listing 2: Creating the data matrix

	0	1	2	3	4	5	6	7	8	9	...	10294	10295	10296	10297	10298	10299	10300	10301	10302	10303
0	48	49	45	47	49	57	39	42	53	49	...	39	44	40	41	49	42	44	47	46	46
1	34	34	33	32	38	40	39	49	54	57	...	42	44	38	30	37	30	36	37	40	33
2	60	60	62	53	48	51	61	60	71	68	...	27	35	28	33	31	31	37	32	34	34
3	39	44	53	37	61	48	61	45	35	40	...	23	30	36	32	28	32	31	29	26	29
4	63	53	35	36	33	34	31	35	39	43	...	173	169	166	161	158	169	137	41	10	24
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
395	110	109	111	107	116	113	111	111	113	114	...	88	97	94	96	91	94	98	94	88	90
396	113	112	111	113	115	115	111	114	115	114	...	92	86	89	86	91	86	93	87	87	89
397	112	109	116	112	113	113	115	114	114	116	...	85	92	91	84	90	87	88	93	88	92
398	111	114	112	112	110	112	111	114	112	112	...	84	83	82	87	82	84	76	88	86	92
399	110	112	113	109	113	110	114	111	111	115	...	93	98	86	88	82	91	87	92	87	90

400 rows × 10304 columns

Figure 3: Data matrix of the images

In Figure 3, present the data matrix of the images and each row represents each image, and each column represents the pixel values of the respective image.

Figure 4: First five Images in the dataset

```
fig, axes = plt.subplots(40,10,figsize=(9,39),
                         subplot_kw={'xticks':[], 'yticks':[]},
                         gridspec_kw=dict(hspace=0.01, wspace
    =0.01)) # creating the layout
for i, ax in enumerate(axes.flat): # going through the all axeces
    for each image
```

```
5      ax.imshow(faces_matrix.iloc[i].values.reshape(112,92),cmap=plt
       .cm.gray) # reshape the image (1 x 10304) to (112 x 92)
6  plt.show() # show the layout
```

Listing 3: Ploting the all images in one layout

Listing 3 code creates the layout of the 40 images, and Figure 4 shows the first five person's images of the dataset.

Listing 4 code is to define the size of the component that we are going to use for applying PCA to the dataset. By changing the value of `n_components` variable, we can observe the result of the images and how many needs not to distinguish the visual difference.

```
1  faces_pca = PCA(n_components=10) # initialice the component size
       of 10 for PCA
```

Listing 4: Initiate the component size

Listing 5 code applies the PCA to compress data with the respective `n_components` value and plotting the compressed images respectively.

```
1  faces_pca = PCA(n_components=10) # initialice the component size
       of PCA
2  faces_pca.fit(faces_matrix) # Apply PCA to the data set
3  components = faces_pca.transform(faces_matrix)
4  projected = faces_pca.inverse_transform(components) #reconstruct
       the images with given components value
5  fig, axes = plt.subplots(40,10,figsize=(9,39),
6                           subplot_kw={'xticks':[], 'yticks':[]},
7                           gridspec_kw=dict(hspace=0.01, wspace
       =0.01)) # creating the layout
8  for i, ax in enumerate(axes.flat):
9      ax.imshow(projected[i].reshape(112,92),cmap=plt.cm.gray) #
       reshape the image (1 x 10304) to (112 x 92)
10 plt.show() # show the layout
```

Listing 5: Apply PCA to data set with the specific component size and reconstruct again

Figure 5 shows the images of five persons, which are reconstructed by applying ten components to PCA.

Figure 5: Recontructed images using ten components

2.4 Results

As shown in Figure 5, 10 components for compression caused the reconstruction of the images with low resolution. Furthermore, increasing the component size by 50, 100, 200, and 300, respectively, and seeing how images minimize the distinguish of the visual difference from the originals.

Firstly, consider applying 50, 100, 200, and 300 components and the result shown in Figure 6, Figure 7, Figure 8, and Figure 9, respectively.

Figure 6: Recontructed images using 50 components

Figure 7: Recontructed images using 100 components

Figure 8: Recontructed images using 200 components

Figure 9: Recontructed images using 300 components

2.5 Discussion

In this analysis, we observed the image compression technique PCA, reviewing and understanding how the method works and later applying it to compress a dataset of images. The purpose of the whole analysis was to conduct PCA based image compression with Python, apply the few components associated with large eigenvalues and reconstruct faces using only these few components as a compression technique. The reconstructed images with 50 and 100 principal components are very different from those applying 200 and 300 principal components. Performing PCA applying the large component size would cause to have a successful output. Applying 200 and 300 components close to the dataset's size (data matrix) has the output of the images that do not distinguish the visual difference from the original image.

In conclusion, as we discussed, the most common compression method used for natural images: is the PCA. This method uses the natural characteristics of the matrices to achieve their goals. The use of linear algebra is very suited to the task of compression. Furthermore, PCA is an iterative optimization problem designed to compress high-dimensional data into fewer dimensions and to minimize the resulting reconstruction loss. The methods developed take advantage of the accuracy and speed of this process and reduce the computer's memory.

3 Problem 2 - Housing price prediction/estimation

3.1 Problem statement

In this section, we will determine the demand price of a house given a collection of features. Any house's market price (The target variable (y)) depends on many factors, such as location, view, and surrounding. Therefore, analysing and predicting housing prices using attributes or features is the primary purpose of this section. In order to achieve that, the linear regression algorithm can be used.

3.2 Background

3.2.1 The least squares method for over-determined systems

In a linear regression model, the goal is to find the line of best fit that describes the relationship between a dependent variable and one or more independent variables. The least squares method is a common mathematical optimization technique used to find the best-fit line (or curve) for data points to identify the connection between a dependent variable and one or more independent variables.

In an overdetermined system, there are more equations (or data points) than unknowns (or variables). In other words, the model has more data points than variables. Therefore, finding a unique solution that satisfies all of the equations simultaneously in the system of equations is impossible, as there will be an infinite number of possible solutions. However, the least squares method can be used to find the "best" solution because it minimizes the summation of the squared differences between the predicted and the observed values.

In order to find the line of best fit using the least squares method, first define a loss function that measures the error between the predicted and observed values. The loss function is typically defined as the sum of the squared differences between predicted and observed values. The solution that minimizes the sum of squares of differences between the observed data and the predicted values identify as the "best" line for the model (The goal is to find the values of the coefficients (or parameters) of the fitted line that minimize the sum of the squares values). The resulting solution is called the least squares solution, which provides an excellent approximation to the proper solution in many cases. It is widely used in statistical analysis and data modelling, particularly in linear regression, where it fits a linear model to a set of data points.

The least squares method finds the best-fit line or curve for data points. It is often used in linear regression to model the connection between a dependent variable and one or more independent variables. In addition, the least squares method is widely used in linear regression models because it is relatively simple to implement and can often provide good results. Also, it is advantageous in the case of overdetermined systems, which have more equations than unknowns, as it allows for an excellent approximation of the actual solution by minimizing the sum of the squares of the residuals.

3.2.2 Connection of least squares method to linear regression models

In a linear regression model, the output variable (y) is modelled as a linear function of the input variable (x). The line of best fit is the line that minimizes the summation of the squared differences between the predicted and the observed values. Also, the best-fit line describes the relationship between the input variables (x) and the output variables (y) as a linear equation. This method is called the least squares method.

The line of best fit is described by the equation:

$$y = m * x + c$$

Where y is the output variable, x is the input variable, m is the slope of the line, and c is the y-intercept of the line.

The aim of linear regression is to determine the values of m and c that best describe the relationship between x and y. The least squares method involves finding the values of m and c that minimize the sum of the squared differences between the predicted values (the values on the line of best fit) and the observed values (the actual data points).

To find the values of m and c that minimize the sum of the squared differences, we can use the following formulas:

$$m = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

$$c = \bar{y} - m * \bar{x}$$

Where $\bar{x}$ and $\bar{y}$ are the sample mean of x and y, respectively, and $\sum$ represents the sum over all the data points in the sample.

These formulas are derived using the method of least squares method. Once we have found the values of the slope and y-intercept, we can use them to make predictions about the output variable (y) given a new input variable (x). This is the fundamental connection between the least squares method and linear regression.

3.3 Method

3.3.1 Data Collection

The data set for housing price prediction contains 506 instances (rows) and 13 attributes (columns), both categorical and numerical values for each instance. The attributes are defined as follows:

- CRIM per capita crime rate by town
- ZN proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS proportion of non-retail business acres per town
- CHAS Charles River dummy variable (1 if tract bounds river; 0 otherwise)
- NOX nitric oxides concentration (parts per 10 million)
- RM average number of rooms per dwelling
- AGE proportion of owner-occupied units built prior to 1940
- DIS weighted distances to five employment centres
- RAD index of accessibility to radial highways
- TAX full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- LSTAT percentage of the lower status of the population
- target Median value of owner-occupied homes in $1000's

3.3.2 Determine the linear formula for price calculations of houses of given data-set

In this phase, we will determine the linear formula for the housing price using the given data set by performing a linear regression algorithm in Python. The solution linear formula will be in the following format:

$$y = a_0 + a_1x_1 + a_2x_2 + ... + a_Nx_N$$

In order to determine the linear relationship between the prices of the houses (target variable) and the other independent variables, use the few built-in libraries in Python.

Before determining the linear relationship, we first need to wrangle the data set to identify the data types of the attributes. Also, it is essential to cleanse the data as well. Below, Python codes show the steps performed in the above mentioned process.

```
import pandas as pd # for data processing to create data frames
    for the dataset (e.g. pd.DataFrame)
import numpy as np # for linear algebra
import matplotlib.pyplot as plt # to display the data set and the
    necessary plots
from sklearn.linear_model import LinearRegression # this is for
    applying and determining the linear relationship among the
    variables
```

Listing 6: Importing the necessary libraries

After importing the libraries shown as Listing 6, the data set is converted to a data frame in panda in Python as Listing 7.

```
# Get the data file 'Problem2_Datas to pandas data frame
df = pd.read_csv('Problem2_Dataset.csv')
df # show the data frame
```

Listing 7: Data set convert to pandas data frame

	Unnamed: 0	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	target
0	0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	4.98	24.0
1	1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	9.14	21.6
2	2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	4.03	34.7
3	3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	2.94	33.4
4	4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	9.67	22.4
502	502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	9.08	20.6
503	503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	5.64	23.9
504	504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	6.48	22.0
505	505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	7.88	11.9

506 rows × 14 columns

Figure 10: Structure of the data frame

Figure 10 shows the structure of the data frame 'df'. Figure 11 is the final data set that we are going to use after dropping the first column 'Unnamed' by executing the code shows in Listing 8.

```
# remove the 'Unnamed' column
df = df.drop(df.columns[[0]], axis = 1)
df # show the data frame
```

Listing 8: Remove the first column

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT	target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	9.67	22.4
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	9.08	20.6
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	5.64	23.9
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	6.48	22.0
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	7.88	11.9

Figure 11: The Final Data set

Listing 9 code checking the missing values of the data set and the data set contains no missing values.

```
# Identify the missing values of the data frame
missing_values_count = df.isnull().sum()
missing_values_count # shows the missing value count to the
   respective column
```

Listing 9: Checking the missing values

Result shows in below Figure 12.

```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
LSTAT      0
target     0
dtype: int64
```

Figure 12: No missing values contain in all attributes

```
df.nunique() # Get the idea of categorical variables
```

Listing 10: Cheking the unique values in all attributes

The next step is to see how many unique values are contained in each attributes to identify the categorical variables. Code in Listing 10 and results are shown in Figure 13.

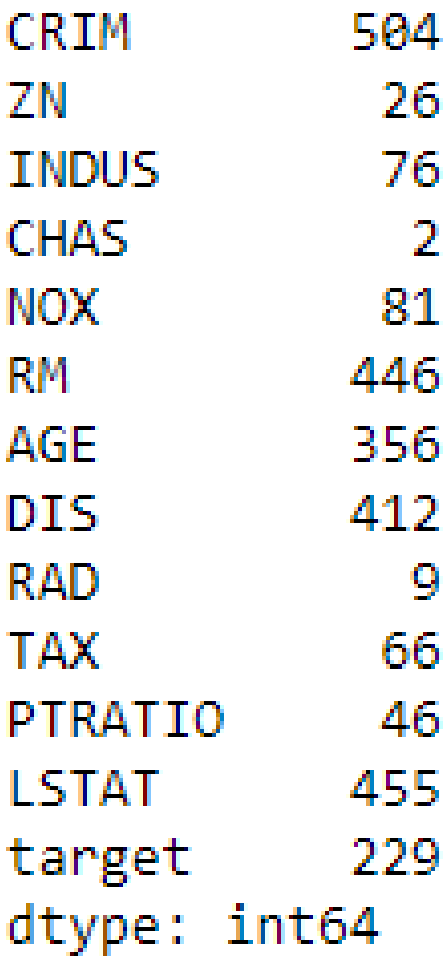


Figure 13: Unique values count in all attributes

```
# Creating the scatter plots to identify between the endogenous variable 'target' and all other variables.
# ploting all x variables with y
x_names = df.columns[0:12]      # Get all independent variables names
y_name = df.columns[-1]         # get the dependant variable 'target' names

for i in x_names:               #iterate through all dependent variables to create the plots
    plt.scatter(df[i],df[y_name]) # define x and y axis data
```

```
    plt.xlabel(i)                    # set the x-axis name
    plt.ylabel(y_name)               # set the y axis name
    plt.title("Scatter plot of "+i+" and "+y_name) # set the title
    of the graph
    plt.show() # show the graph
```

Listing 11: Plotting the data

By plotting the graphs, we can understand the data more effectively, how the target variable behaves with the independent variables respectively. Listing 11 shows the code to execute for plotting the data. All the scatter plots show the distribution of the data of the target variable (Prices of the housing) with each independent variable Figure 14.

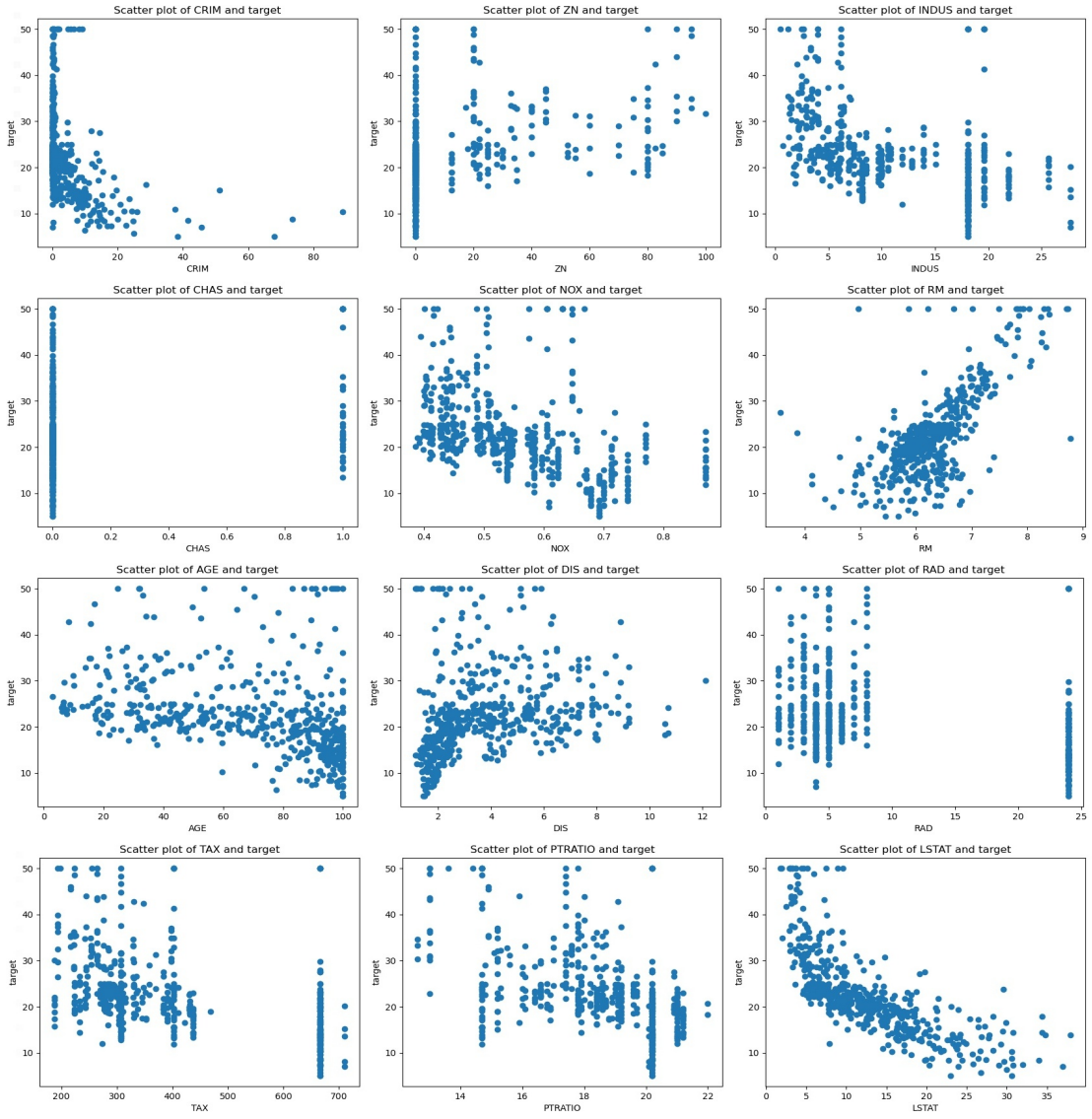


Figure 14: Plots of all attributes against target

Figure 14 (plots) shows the relation between a target variable (target-Price) and all other independent variables. By referring to plots and the unique values (Figure 13), it is evident that most variables are biased; many have outliers. Variables ZN, CHAS, RAD, and TAX seem categorical as they have only several specified values. Also, from the data-set description, only CHAS is categorical as it gives information if the home bounds tract with Charles river. Variables RM and LSTAT show the best correlation based on these plots. RM corresponds to the number of rooms per dwelling and positively affects the house price; this is not surprising as bigger houses cost more money than smaller ones. LSTAT, on the other hand, show a negative relationship which is also not surprising as it corresponds to the percentage of the lower-status population.

However, in the next step, we will calculate the coefficients of all independent variables using a regression model in Python Listing 12.

```
# Finding multilinear coefficients for all independent variables

x_data = df.iloc[:,0:12]  # get the data of indipendant variables
y_data = df.iloc[:,12:13] # get the data of target variables

model = LinearRegression() # defing the linear regression model
X = np.asanyarray(x_data)  # get the independent variables data to
    array
Y = np.asanyarray(y_data)  # get the dependent variable (target)
   data to array

model.fit (X, Y)           # fit the data to the regression model

coefficients = pd.DataFrame(model.coef_, index=['Coefficients'],
   columns=x_names) # get the coefficient values to a data frame

print ('Intercept: ', model.intercept_) # print the value of the
   intercept
coefficients   # print the value of coefficients
```

Listing 12: Finding the coefficients

The result of code in Listing 12. show in Figure 15.

Intercept: [41.61727018]

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	LSTAT
Coefficients	-0.121389	0.046963	0.013468	2.839993	-18.758022	3.658119	0.003611	-1.490754	0.289405	-0.012682	-0.937533	-0.552019

Figure 15: Intercept and Coefficients of the linear formula

Determine the linear formula using all independent variables is unsuitable because some variables have categorical characteristics. We can prove that by finding the mean square error for each independent variable shown in Figure 16. this result originated from the code shown in Listing 13.

```
# Calculate the sum of squares using the least squares method
model = LinearRegression() # defing the linear regression model
temp = [] # defing a list to store the data
for col in x_data.columns: # for loop iterates through the
    independent variables
     X = np.asanyarray(x_data[[col]]) # get the one independent
    variable
     Y = np.asanyarray(y_data)        # get the dependent variable
     model.fit (X, Y)                 # fit the data to the
    regression model
     y_hat= model.predict(x_data[[col]]) # get the values for new
    target price define by model
     square_error = np.mean((y_hat - Y) ** 2) # calculating the
    mean square error
     lst = [col, model.coef_, model.intercept_, square_error] # a
    list to store data
     temp.append(lst)
table = pd.DataFrame(temp, columns=['Independent Variable', '
    Coefficient', 'Intercept', 'Square Error']) # get the data to a
     data frame
table.sort_values(by ='Square Error', ascending=True, ignore_index
     = True) # show the data frame with sorted values by mean
    square error
```

Listing 13: Finding the Mean square error for all independent variables

	Indipendent Variable	Coefficient	Intercept	Square Error
0	LSTAT	[[-0.9500493537579909]]	[34.55384087938311]	38.482967
1	RM	[[9.102108981180308]]	[-34.670620776438554]	43.600552
2	PTRATIO	[[-2.157175296060966]]	[62.34462747483268]	62.652200
3	INDUS	[[-0.6484900536157153]]	[29.754896511928493]	64.666222
4	TAX	[[-0.025568099481987277]]	[32.9706544936663]	65.887275
5	NOX	[[-33.916055008661104]]	[41.345874467973246]	69.004288
6	CRIM	[[-0.4151902779150907]]	[24.03310617412388]	71.690736
7	RAD	[[-0.403095395552531]]	[26.382128362272397]	72.124812
8	AGE	[[-0.12316272123567969]]	[30.97867776261804]	72.423981
9	ZN	[[0.1421399941553544]]	[20.917579117799832]	73.451696
10	DIS	[[1.0916130158411093]]	[18.390088330493384]	79.146342
11	CHAS	[[6.3461571125265355]]	[22.093842887473468]	81.826514

Figure 16: Intercept, Coefficients, and Mean square error of each variable

As shown in Figure 16, it is evident that LSTAT and RM variables have the lowest square error (below fifty) for the target variable Price of the housing. Therefore, defining the linear relationship using only those two independent variables is suitable. In order to do that, we require to find the respective coefficients of those two variables. Listing 14 code is to find the coefficients, and Figure 17 shows the results of the code in Listing 14.

```
# Calculate the coefficient of LSTAT and RM independent variables
model = LinearRegression() # defing the linear regression model

X = np.asanyarray(x_data[['LSTAT','RM']]) # get the LSTAT and RM
    independent variables data to array
Y = np.asanyarray(y_data) # get the dependent variable (target)
    data to array

model.fit (X, Y) # fit the data to the regression model

coefficients = pd.DataFrame(model.coef_, index=['Coefficients'],
    columns=['LSTAT','RM']) # get the coefficient values to a data
    frame

```

```
print ('Intercept: ', model.intercept_) # print the value of
    intercept
coefficients                             # print the value of
    coefficients
```

Listing 14: Calculating the coefficient of LSTAT and RM

```
Intercept:   [-1.35827281]
```

	LSTAT	RM
Coefficients	-0.642358	5.094788

Figure 17: Intercept and Coefficients of LSTAT and RM variable

3.4 Results

As we discussed in the method section, it is essential to identify the linearity of the variable to the target variable. According to Figure 15, the intercept and coefficients of all variables are as follows:

- Intercept - a_0 - 41.61727018
- CRIM - a_1 - (-0.121389)
- ZN - a_2 - 0.046963
- INDUS - a_3 - 0.013468
- CHAS - a_4 - 2.839993
- NOX - a_5 - (-18.758022)
- RM - a_6 - 3.658119

- AGE - a_7 - 0.003611
- DIS - a_8 - (-1.490754)
- RAD - a_9 - 0.289405
- TAX - a_{10} - (-0.012682)
- PTRATIO - a_{11} - (-0.937533)
- LSTAT - a_{12} - (-0.552019)

By using these twelve coefficients and intercept, we can define the linear formula of price calculations as follows:

$$Price(y) = 41.61727018 + (-0.121389) * CRIM + 0.046963 * ZN + 0.013468 * INDUS + 2.839993 * CHAS + (-18.758022) * NOX + 3.658119 * RM + 0.003611 * AGE + (-1.490754) * DIS + 0.289405 * RAD + (-0.012682) * TAX + (-0.937533) * PTRATIO + (-0.552019) * LSTAT$$

However, we found out by observing the plots and the mean square errors that the most linear variables to the target variable are RM and LSTAT. Therefore, the most suitable way to define the linear formula of price calculations is to consider these two variables and their respective intercept. Figure 17 shows the respective coefficient and intercept for those two variables as follows:

- Intercept - b_0 - (-1.35827281)
- RM - b_1 - 5.094788
- LSTAT - b_2 - (-0.642358)

Finally, we can define a proper linear formula for price calculation of the houses considering those two variables as follows:

$$Price(y) = (-1.35827281) + 5.094788 * RM + (-0.642358) * LSTAT$$

3.5 Discussion

In conclusion, the linear relationship between the variables depends on how they actually linear with each other. For instance, as we discussed the price prediction of houses, there were only two variables in line with the linearity of the target variable.

The least-squares method is sensitive to outliers. This could cause problems with the results of the least-squares analysis.

Furthermore, considering the limitation of the least square method, it is most suitable to find the linear relationship between only two variables. Because this method represents only the relationship between two variables, all other factors, such as causes and effects, are not considered. Also, the method is unreliable when data is not evenly distributed throughout the data, which means the method is accurate only if the data genuinely fall on a straight line. That is the reason we have the highest mean square errors for most independent variables in the case of price calculation.

References

[1] Steven L. Brunton & J. Nathan Kutz, "Data Driven Science & Engineering," *2017, Machine Learning, Dynamical Systems, and Control*, 2017.

[2] Sunny Verma & Jakkam Phanindra Krishna, "Image Compression and Linear Algebra," , 2013.

[3] Aidan Meacham, "MATH420 Project - Image Compression," , 2014.

[4] GeeksforGeeks,"https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/," , 2019.

[5] Zichen Wang, "https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8," , 2019.

[6] Nalin Gadihoke, "https://www.nalingadihoke.com/post/linalg/," , 2020.

[7] Philipp Wagner, "https://www.bytefish.de/blog/eigenfaces.html," , 2011.

[8] Adrian Tam, "https://machinelearningmastery.com/face-recognition-using-principal-component-analysis/," , 2021.

[9] Sebastian Norena"https://medium.com/@sebastiannorena/pca-principal-components-analysis-applied-to-images-of-faces-d2fc2c083371," , 2018.

[10] Hua Shi, "https://melaniesoek0120.medium.com/principal-component-analysis-pca-facial-recognition-5e1021f55151," , 2020.

[11] Stack Exchange Inc, "https://math.stackexchange.com/questions/3500898/understanding-the-least-squares-regression-formula," , 2023.

[12] Cuemath, "https://www.cuemath.com/data/least-squares/," , 2022.

[13] CYNTHIA HELZNER, "https://study.com/academy/lesson/least-squares-regression-definition-equations-examples.html," , 2022.

[14] Overleaf, "https://www.overleaf.com,"

[15] pandas via NumFOCUS, Inc, "https://pandas.pydata.org/docs/reference/frame.html," , 2022.

[16] NumPy Developers, "https://numpy.org/doc/stable/user/index.html," , 2008-2022.

[17] Matplotlib development team, "https://matplotlib.org/stable/tutorials/introductory/pyplot.html," , 2012-2023.

[18] scikit-learn developers, "https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html," , 2007-2022.

[19] scikit-learn developers, "https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html," , 2007-2022.