Dataset: https://www.kaggle.com/datasets/maharshipandya/-cryptocurrency-historical-prices-dataset/code?resource=download

Write a R program to Apply standardisation/ normalisation techniques (create min. 3 types in each category) on the given dataset.

```
# Load the dataset
library(MASS)
library(dplyr)
library(readr)
crypto <- read.csv("/content/dataset.csv")
```

```
Attaching package: ‘dplyr’


The following object is masked from ‘package:MASS’:

    select


The following objects are masked from ‘package:stats’:

    filter, lag


The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union
```

```
head(crypto)
```

A data.frame: 6 × 10

	X	open	high	low	close	volume	marketCap	
	<int>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	
1	0	112.90000	118.80000	107.14300	115.91000	0	1288693176	05
2	1	3.49313	3.69246	3.34606	3.59089	0	62298185	05
3	2	115.98000	124.66300	106.64000	112.30000	0	1249023060	06

```
str(crypto)
```

```
'data.frame':   72946 obs. of  10 variables:
 $ X          : int  0 1 2 3 4 5 6 7 8 9 ...
 $ open       : num  112.9 3.49 115.98 3.59 112.25 ...
 $ high       : num  118.8 3.69 124.66 3.78 113.44 ...
 $ low        : num  107.14 3.35 106.64 3.12 97.7 ...
 $ close      : num  115.91 3.59 112.3 3.37 111.5 ...
 $ volume     : num  0 0 0 0 0 0 0 0 0 0 ...
 $ marketCap  : num  1.29e+09 6.23e+07 1.25e+09 5.86e+07 1.24e+09 ...
 $ timestamp  : chr  "2013-05-05T23:59:59.999Z" "2013-05-05T23:59:59.999Z" "2013-05-06T23:59:59.999Z" "2013-05-06T23:59:59.999Z" ...
 $ crypto_name: chr  "Bitcoin" "Litecoin" "Bitcoin" "Litecoin" ...
 $ date       : chr  "2013-05-05" "2013-05-05" "2013-05-06" "2013-05-06" ...
```

Standardization

Standardization is a process of transforming a dataset so that it has a mean of 0 and a standard deviation of 1. This is done by subtracting the mean of the data from each value and dividing by the standard deviation. The purpose of standardization is to scale the data to a common level so that each feature is treated equally and so that the data can be more easily compared and analyzed. By standardizing the data, it also ensures that extreme values or outliers will not have a disproportionate impact on the results of any analysis or modeling performed on the data. In short, standardization is a way to normalize the data so that it has a standard, well-defined format.

```
# Label encode the categorical variables
crypto_categorical <- crypto[c("timestamp","crypto_name", "date")]
crypto_categorical$timestamp <- as.numeric(as.factor(crypto_categorical$timestamp))
crypto_categorical$crypto_name <- as.numeric(as.factor(crypto_categorical$crypto_name))
crypto_categorical$date <- as.numeric(as.factor(crypto_categorical$date))

# Standardize the numerical variables
crypto_numeric <- crypto[c("open", "high", "low", "close","volume","marketCap")]

new_crypto <- cbind(crypto_numeric,crypto_categorical)
head(new_crypto)
str(new_crypto)
```

A data.frame: 6 × 9

	open	high	low	close	volume	marketCap	timestamp
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	112.90000	118.80000	107.14300	115.91000	0	1288693176	1
2	3.49313	3.69246	3.34606	3.59089	0	62298185	1
3	115.98000	124.66300	106.64000	112.30000	0	1249023060	2
4	3.59422	3.78102	3.11602	3.37125	0	58594361	2
5	112.25000	113.44400	97.70000	111.50000	0	1240593600	3
6	3.37087	3.40672	2.93979	3.33274	0	58051265	3

```
'data.frame':	72946 obs. of  9 variables:
 $ open      : num  112.9 3.49 115.98 3.59 112.25 ...
 $ high      : num  118.8 3.69 124.66 3.78 113.44 ...
 $ low       : num  107.14 3.35 106.64 3.12 97.7 ...
 $ close     : num  115.91 3.59 112.3 3.37 111.5 ...
 $ volume    : num  0 0 0 0 0 0 0 0 0 0 ...
 $ marketCap : num  1.29e+09 6.23e+07 1.25e+09 5.86e+07 1.24e+09 ...
 $ timestamp : num  1 1 2 2 3 3 4 4 5 5
```

```
tail(crypto_categorical)
```

A data.frame: 6 × 3

	timestamp	crypto_name	date
	<dbl>	<dbl>	<dbl>
72941	3248	49	3248
72942	3248	54	3248
72943	3248	26	3248
72944	3248	25	3248
72945	3248	43	3248
72946	3248	56	3248

```
# Z-Score Standardization
z_score_standardize <- function(x) {
  return ((x - mean(x)) / sd(x))
}
crypto_stand_zscore <- as.data.frame(lapply(new_crypto[2:ncol(new_crypto)], z_score_standardize))
head(crypto_stand_zscore)

# Range Standardization
range_standardize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
crypto_stand_range <- as.data.frame(lapply(new_crypto[2:ncol(new_crypto)], range_standardize))
head(crypto_stand_range)
# Unit Vector Standardization
unit_vector_standardize <- function(x) {
  return (x / sqrt(sum(x^2)))
}
crypto_stand_unit_vector <- as.data.frame(lapply(new_crypto[2:ncol(new_crypto)], unit_vector_standardize))
head(crypto_stand_unit_vector)
```

A data.frame: 6 × 8

	high	low	close	volume	marketCap	timestamp	crypto_name	date
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	-0.1440393	-0.1450800	-0.1442811	-0.2295315	-0.1794460	-3.225581	-1.39353085	-3.225581
2	-0.1653610	-0.1655149	-0.1657344	-0.2295315	-0.1957954	-3.225581	0.08869081	-3.225581
3	-0.1429533	-0.1451790	-0.1449706	-0.2295315	-0.1799748	-3.224200	-1.39353085	-3.224200
4	-0.1653446	-0.1655602	-0.1657764	-0.2295315	-0.1958448	-3.224200	0.08869081	-3.224200
5	-0.1450314	-0.1469390	-0.1451234	-0.2295315	-0.1800872	-3.222820	-1.39353085	-3.222820
6	-0.1654139	-0.1655949	-0.1657837	-0.2295315	-0.1958520	-3.222820	0.08869081	-3.222820

A data.frame: 6 × 8

	high	low	close	volume	marketCap	timestamp	crypto_name	date
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	7.324822e-04	1.612174e-03	1.715487e-03	0	1.010873e-03	0.0000000000	0.1272727	0.0000000000
2	2.276651e-05	5.034794e-05	5.314575e-05	0	4.886778e-05	0.0000000000	0.5454545	0.0000000000
3	7.686315e-04	1.604605e-03	1.662058e-03	0	9.797554e-04	0.0003079766	0.1272727	0.0003079766
4	2.331254e-05	4.688655e-05	4.989504e-05	0	4.596244e-05	0.0003079766	0.5454545	0.0003079766
5	6.994588e-04	1.470085e-03	1.650218e-03	0	9.731432e-04	0.0006159532	0.1272727	0.0006159532
6	2.100473e-05	4.423482e-05	4.932509e-05	0	4.553642e-05	0.0006159532	0.5454545	0.0006159532

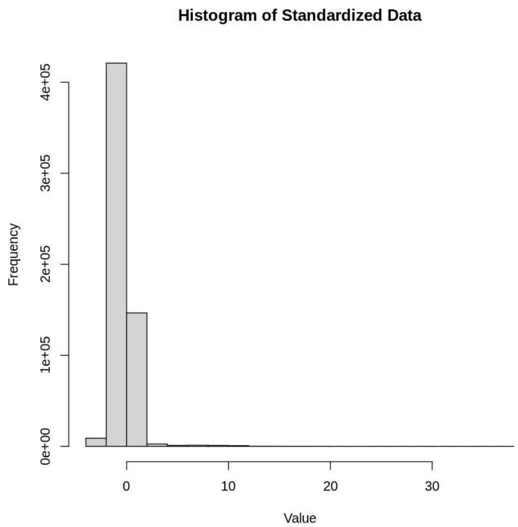
A data.frame: 6 × 8

	high	low	close	volume	marketCap	timestamp	crypto_name	date
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	8.037677e-05	7.704412e-05	8.085967e-05	0	6.241460e-05	1.513096e-06	0.0008857284	1.513096e-06
2	2.498215e-06	2.406077e-06	2.505031e-06	0	3.017255e-06	1.513096e-06	0.0034321976	1.513096e-06
3	8.434351e-05	7.668243e-05	7.834131e-05	0	6.049328e-05	3.026193e-06	0.0008857284	3.026193e-06
4	2.558133e-06	2.240660e-06	2.351809e-06	0	2.837870e-06	3.026193e-06	0.0034321976	3.026193e-06
5	7.675305e-05	7.025387e-05	7.778322e-05	0	6.008502e-05	4.539289e-06	0.0008857284	4.539289e-06
6	2.304892e-06	2.113937e-06	2.324944e-06	0	2.811567e-06	4.539289e-06	0.0034321976	4.539289e-06

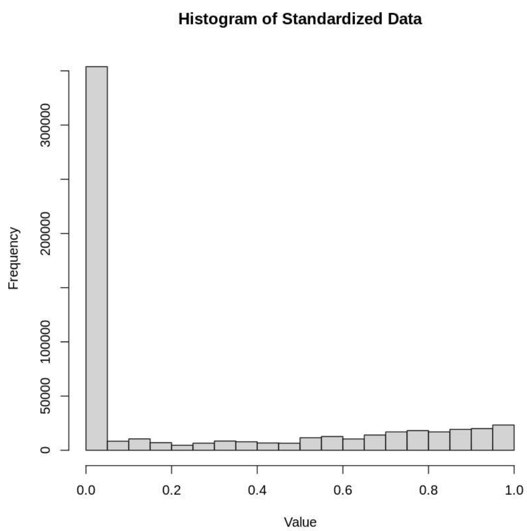
To check the results of normalization, you can use summary statistics such as mean and standard deviation. After normalizing the data, the mean should be close to 0 and the standard deviation should be close to 1. Additionally, you can also visualize the distribution of the data using histograms or density plots. By doing this, you can confirm that the data has been normalized as expected.

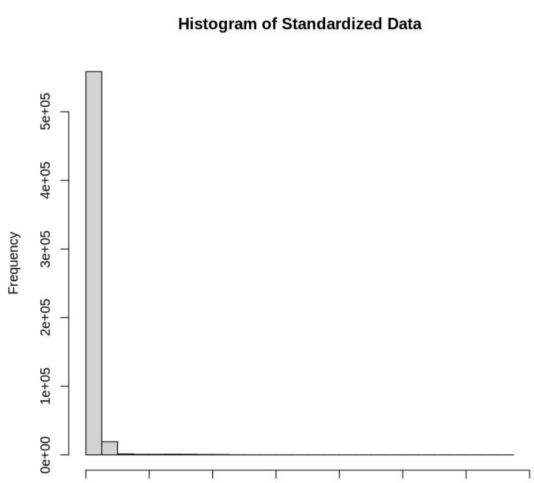
```
# Z-Score Standardization
check_standardization <- function(stand_data) {
  # Check mean
  stand_data = unlist(stand_data)
  mean_stand_data <- mean(stand_data)
  print(sprintf("Mean of standardized data: %f", mean_stand_data))
  # Check standard deviation
  sd_stand_data <- sd(stand_data)
  print(sprintf("Standard deviation of standardized data: %f", sd_stand_data))
  # Check distribution
  hist(stand_data, main = "Histogram of Standardized Data", xlab = "Value")
}

# Example usage
# Check Z-Score Standardization
check_standardization(crypto_stand_zscore)

# Check Range Standardization
check_standardization(crypto_stand_range)

# Check Unit Vector Standardization
check_standardization(crypto_stand_unit_vector)
```

```
[1] "Mean of standardized data: 0.000000"
[1] "Standard deviation of standardized data: 0.999994"
[1] "Mean of standardized data: 0.251061"
[1] "Standard deviation of standardized data: 0.353356"
```




```
[1] "Mean of standardized data: 0.001715"
[1] "Standard deviation of standardized data: 0.003282"
```





Z-Score Standardization: Z-Score standardization is a method of transforming data by subtracting the mean of the data and then dividing by the standard deviation. The result is a "Z-score" for each data point, which represents how many standard deviations the value is from the mean. This method is useful for normalizing the scale of different features so that they can be compared more easily.

Range Standardization: Range standardization is a method of transforming data by scaling it so that it fits within a specific range, usually [0,1]. To perform range standardization, you subtract the minimum value in the data from each value and then divide by the range (i.e. the difference between the maximum and minimum values). This method is useful when the scale of the data is important and you want to control the range of values that are produced.

Unit Vector Standardization: Unit vector standardization is a method of transforming data by dividing each value by the magnitude of the vector that represents the data. The result is a "unit vector" that has a magnitude of 1. This method is useful for normalizing the scale of different features so that they can be compared more easily. It can also be used to project data onto a unit sphere, which can be useful for certain machine learning algorithms.

Normalization

Normalization is a process in which the values of a dataset are transformed to a common scale in order to make comparisons and analysis easier. This is often done by adjusting the values so that they fall within a specific range, such as between 0 and 1. The goal of normalization is to remove any inherent biases in the data and ensure that each feature is treated equally when building models or performing analysis. Normalization is commonly used in data science and machine learning to prepare data for further processing and analysis.

```
# Min-Max Normalization
min_max_normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
crypto_norm_minmax <- as.data.frame(lapply(new_crypto[2:ncol(new_crypto)], min_max_normalize))
head(crypto_norm_minmax)
# Decimal Scaling Normalization
decimal_scaling_normalize <- function(x) {
  k <- max(abs(x))
  return (x / (10^floor(log10(k))))
}
crypto_norm_decimal <- as.data.frame(lapply(new_crypto[2:ncol(new_crypto)], decimal_scaling_normalize))
head(crypto_norm_decimal)
# Log Transformation Normalization
log_transform_normalize <- function(x) {
  return (log10(x + 1))
}
crypto_norm_log <- as.data.frame(lapply(new_crypto[2:ncol(new_crypto)], log_transform_normalize))
head(crypto_norm_log)
```

A data.frame: 6 × 8

	high	low	close	volume	marketCap	timestamp	crypto_name	date
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	7.324822e-04	1.612174e-03	1.715487e-03	0	1.010873e-03	0.0000000000	0.1272727	0.0000000000
2	2.276651e-05	5.034794e-05	5.314575e-05	0	4.886778e-05	0.0000000000	0.5454545	0.0000000000

Min-Max Normalization: Min-Max normalization is a method of transforming data by scaling it so that it fits within a specific range, usually [0,1]. To perform Min-Max normalization, you subtract the minimum value in the data from each value and then divide by the range (i.e. the difference between the maximum and minimum values). This method is useful when the scale of the data is important and you want to control the range of values that are produced.

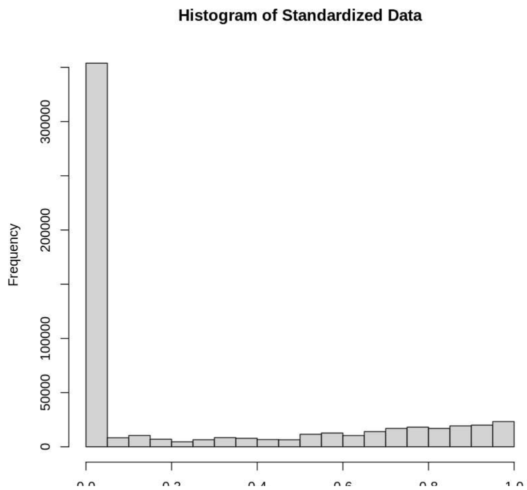
Decimal Scaling Normalization: Decimal scaling normalization is a method of transforming data by dividing each value by a power of 10. The goal of this method is to make the magnitude of the values more manageable by removing unnecessary zeros from the data. For example, if a value is in the millions, dividing by 1,000,000 can help to make it more easily comparable to other values.

Log Transformation: A log transformation is a mathematical operation that transforms data by taking the logarithm of each value. Log transformations are often used to stabilize the variance of the data, making it easier to model and visualize. Log transformations can also help to remove skewness from the data, which can improve the accuracy of certain analyses.

3	1.24663e-03	0.010664000	0.011230000	0	1.249023e-03	0.002	0.8	0.002

```
check_standardization(crypto_norm_minmax)
check_standardization(crypto_norm_decimal)
check_standardization(crypto_norm_log)
```

```
[1] "Mean of standardized data: 0.251061"
[1] "Standard deviation of standardized data: 0.353356"
[1] "Mean of standardized data: 0.981797"
[1] "Standard deviation of standardized data: 1.414790"
```



The checking of the results of standardization and normalization is similar because both techniques aim to transform the data into a standard scale. In standardization, the mean of the data is usually centered at 0 and the standard deviation is usually close to 1. In normalization, the data is scaled between a given range, usually between 0 and 1. In both cases, the summary statistics of the transformed data can be used to confirm that

Inbuilt Functions

R has several inbuilt functions for normalization and standardization of data, some of them are:

scale(): It standardizes the values in a dataset to have a mean of 0 and a standard deviation of 1.

normalize(): It normalizes the values in a dataset to lie within a specified range, often between 0 and 1.

zscore(): It calculates the Z-Score of each value in a dataset, which is the number of standard deviations a value is from the mean.

decimate(): It reduces the number of decimal places of a dataset by rounding values to a specified number of decimal places.

log(): It calculates the logarithm of each value in a dataset to a specified base.

sqrt(): It calculates the square root of each value in a dataset.

minmax(): It normalizes the values in a dataset to lie between 0 and 1, by subtracting the minimum value and dividing by the range.

scale(): The function scale() in R standardizes data by subtracting the mean and dividing by the standard deviation. The formula used is x = (x - mean(x)) / sd(x)

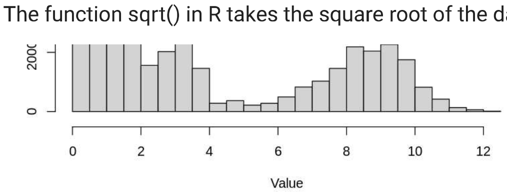
normalize(): The function normalize() in R normalizes data by dividing the data by the norm of the data. The formula used is x = x / sqrt(sum(x^2))

rescale(): The function rescale() in R normalizes data by transforming the data to lie between a specified range, typically 0 and 1. The formula used is x = (x - min(x)) / (max(x) - min(x))

log(): The function log() in R applies the natural logarithm to the data. The formula used is x = log(x)

log10(): The function log10() in R applies the logarithm to the base 10 to the data. The formula used is x = log10(x)

sqrt(): The function sqrt() in R takes the square root of the data. The formula used is x = sqrt(x)



0s completed at 11:53 AM