COMPUTER SCIENCE AND DATA ANALYTICS

Course: Intro to Big Data Analytics

Project 2

Student(s): Emin Alizada, Rustam Talibzade

Instructor: **Dr. Abzatdin Adamov**

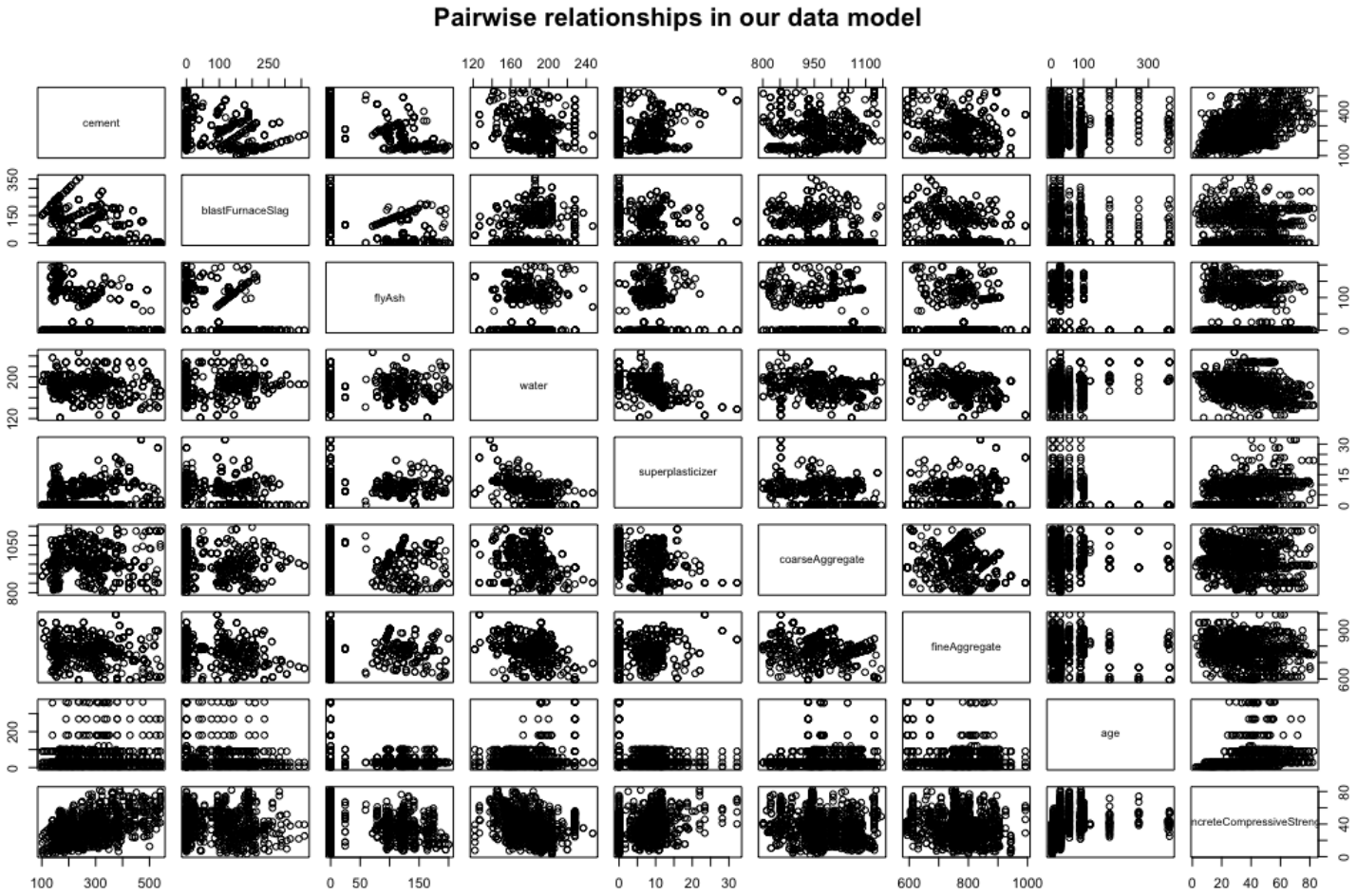
Baku 2021

Table of Contents

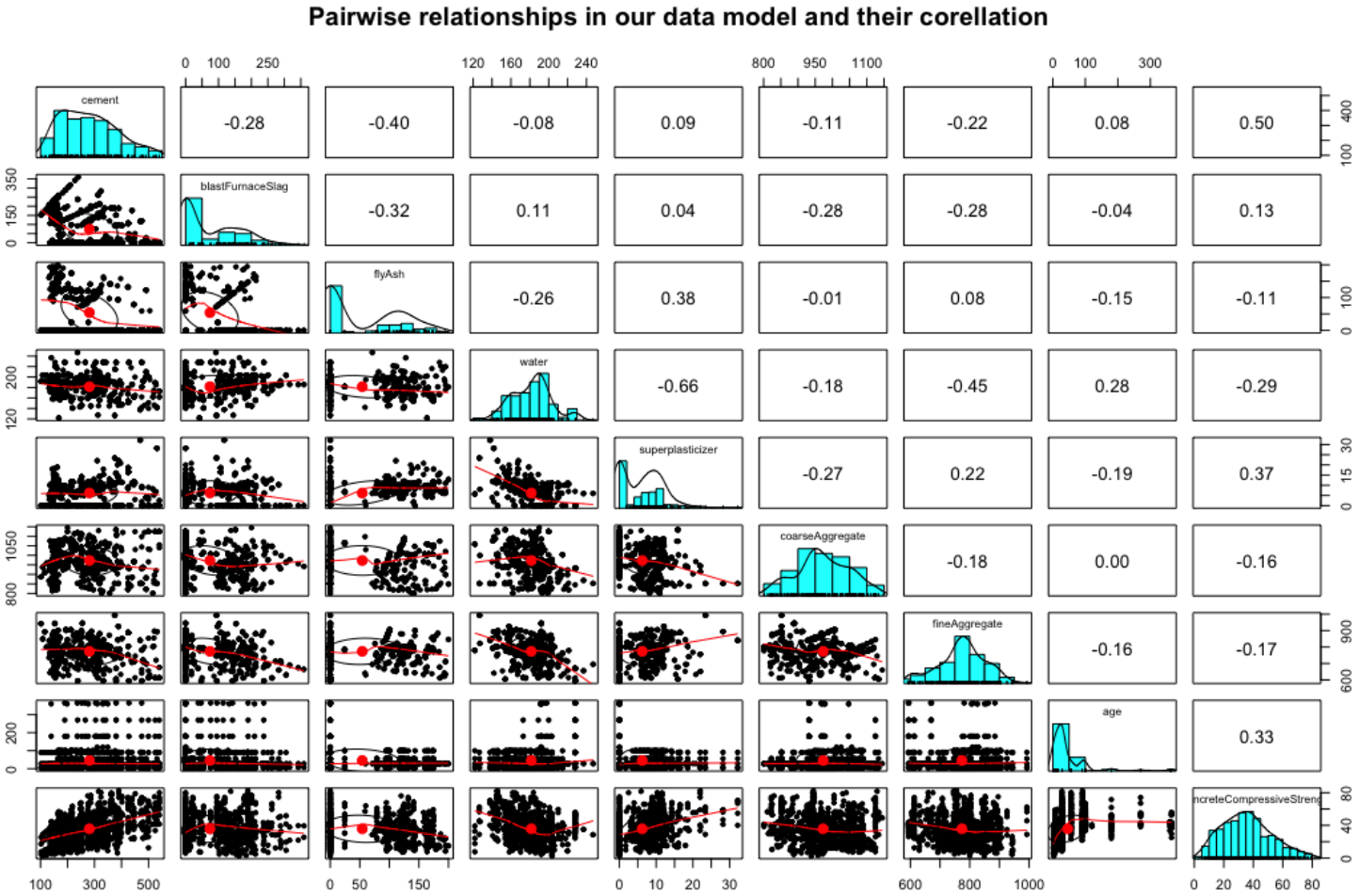
Part 1

We started exploration of our data model by pairs() function, pairs.panel() function and ggpairs() function to see the pairwise relationships between attributes and their correlation:

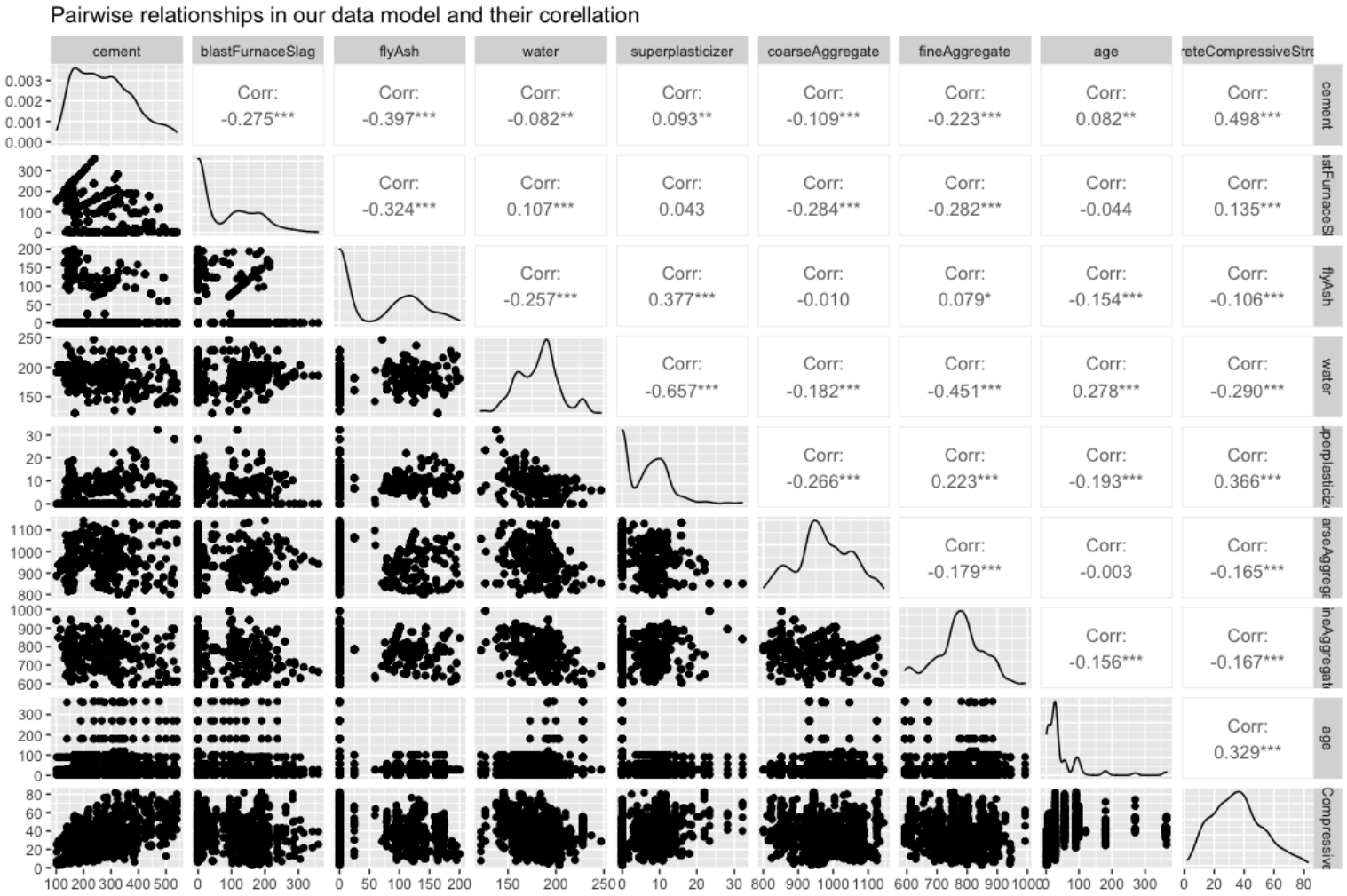
pairs() from standard packages:



pairs.panel() from psych package:

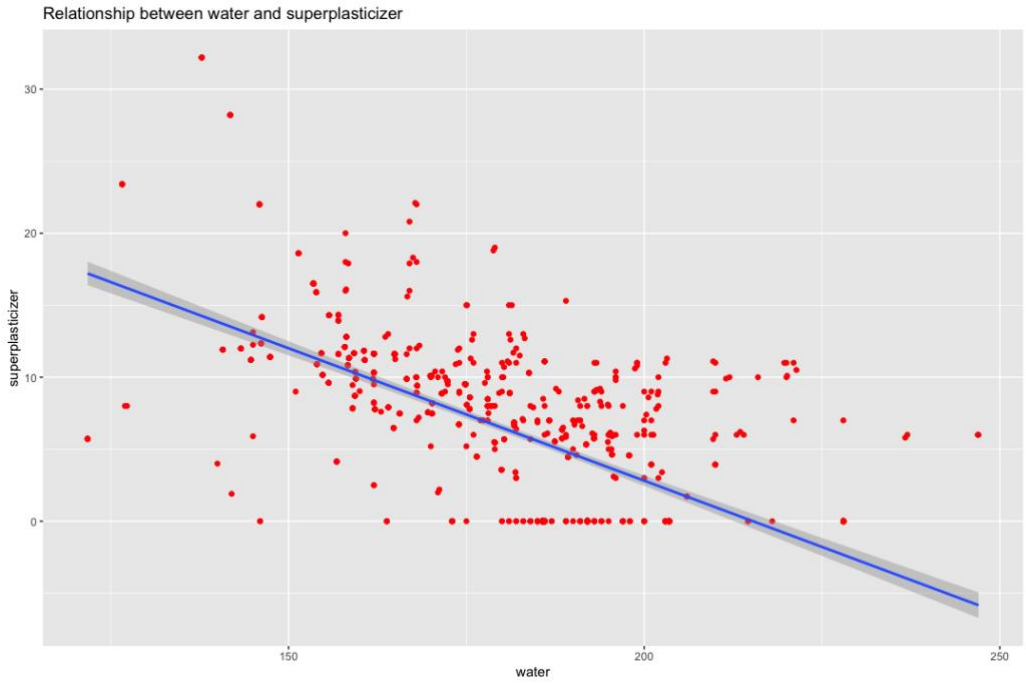


ggpairs() from GGally package:

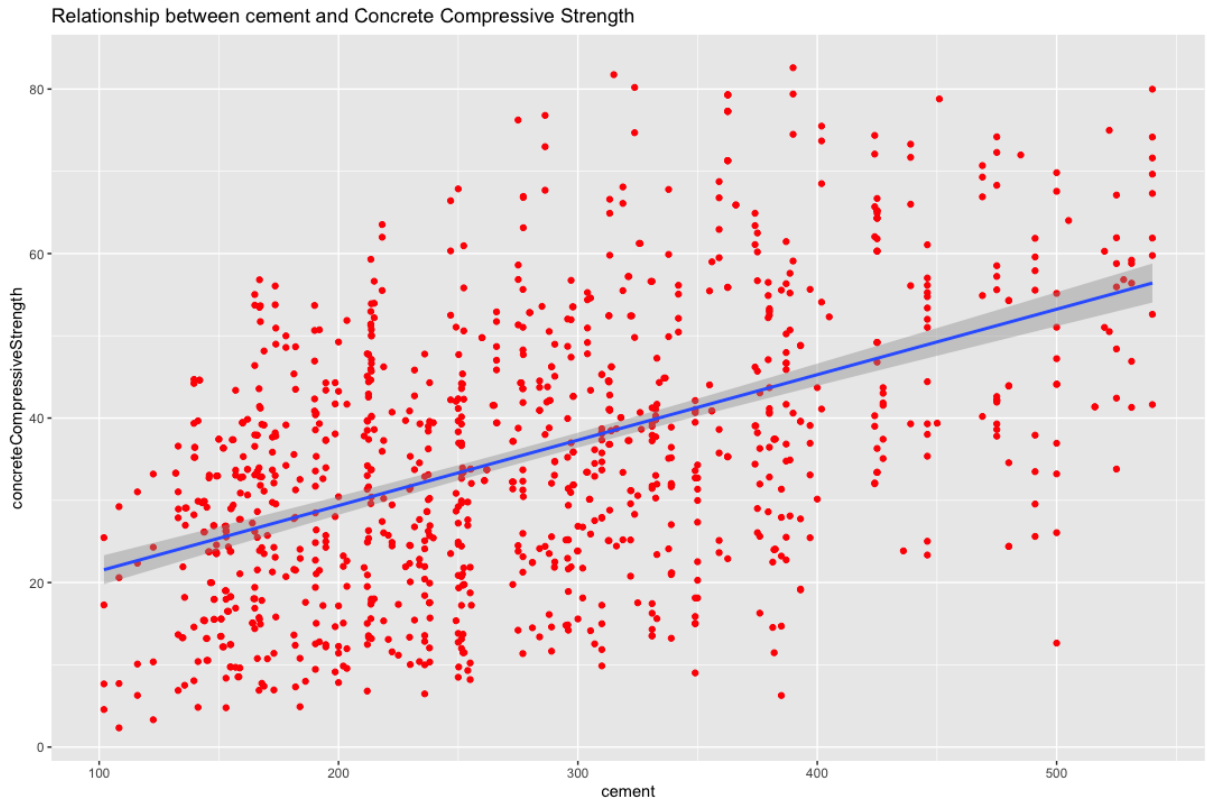


As can be seen from the plots additional packages provide extra functionality while paring attributes, they also show their correlation to each other.

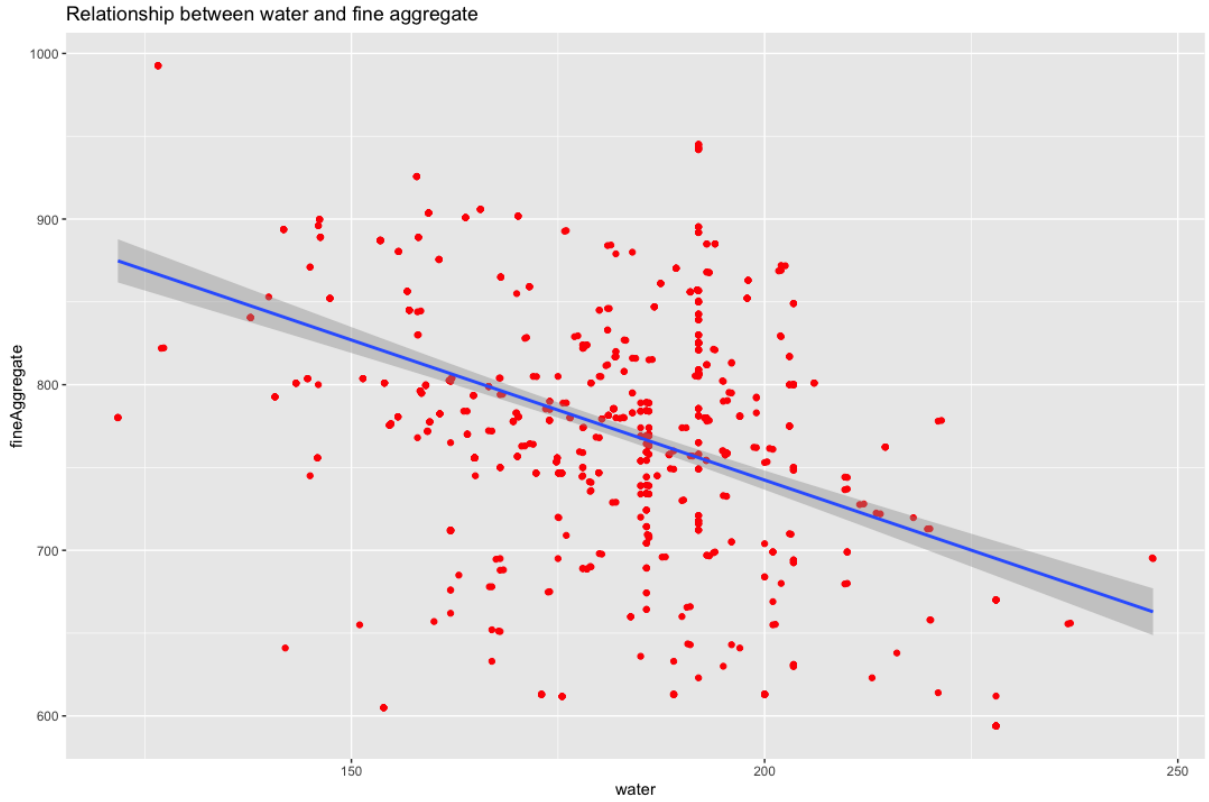
From the produced outputs we can see the biggest correlation between the relationship of superplasticizer and water which is -0.657, it means they have negative trend, so increase in amount of water ends up in decrease in amount of superplasticizer.



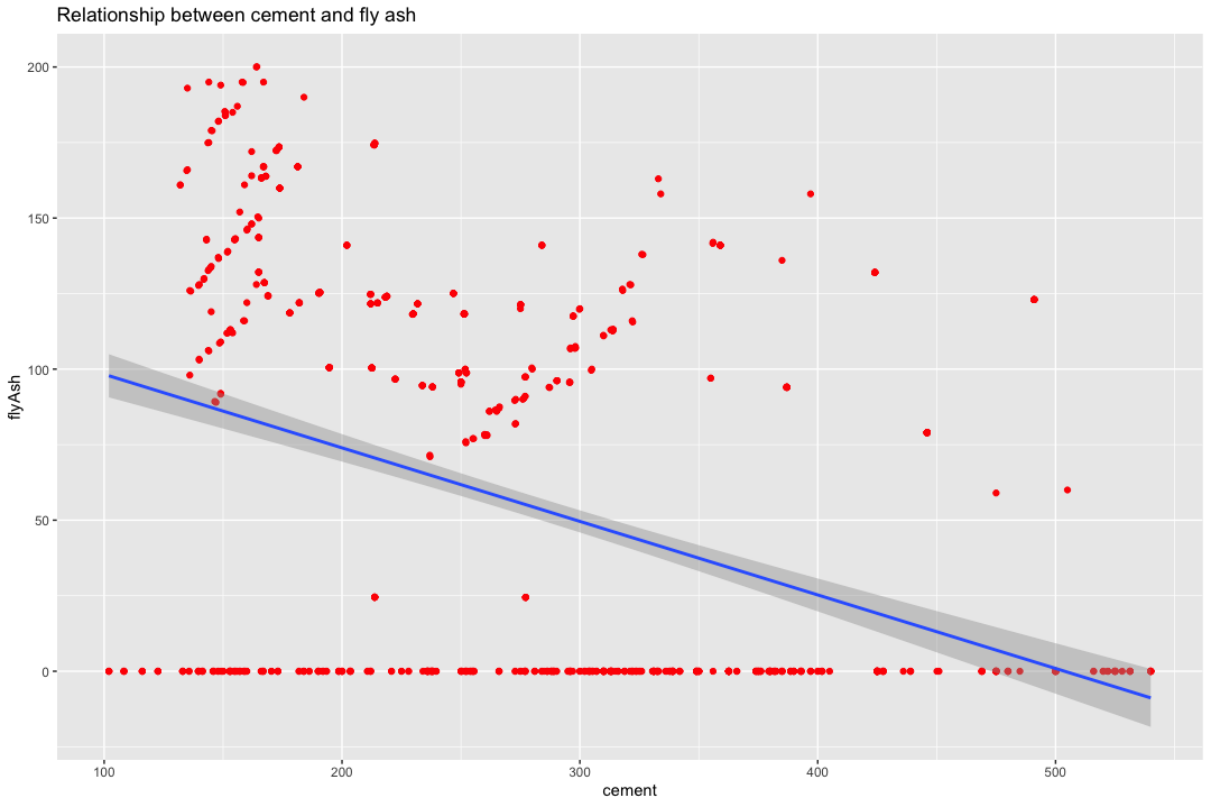
Second biggest correlation is observed between cement and concrete compressive strength which is 0.498, it means they have positive trend, so increase in amount of cement in mixture means increase in concrete compressive strength.



Another interesting relationship is observed between water and fine aggregate, they have a correlation of -0.451. More water means less fine aggregate in the mixture.



Lastly, there is a high correlation between cement and fly ash, which is -0.397. Again there is a negative trend between them, so more cement usually means less fly ash.



Part 2

The first function we have used is str() which shows us structure of an object in R. As can be seen, our data frame has 1030 observations of 9 variables and all variables are numbers.

```
> str(df)
tibble [1,030 × 9] (S3: tbl_df/tbl/data.frame)
 $ cement                      : num [1:1030] 540 540 332 332 199 ...
 $ blastFurnaceSlag            : num [1:1030] 0 0 142 142 132 ...
 $ flyAsh                      : num [1:1030] 0 0 0 0 0 0 0 0 0 0 ...
 $ water                       : num [1:1030] 162 162 228 228 192 228 228 228 228 228 ...
 $ superplasticizer            : num [1:1030] 2.5 2.5 0 0 0 0 0 0 0 0 ...
 $ coarseAggregate             : num [1:1030] 1040 1055 932 932 978 ...
 $ fineAggregate               : num [1:1030] 676 676 594 594 826 ...
 $ age                         : num [1:1030] 28 28 270 365 360 90 365 28 28 28 ...
 $ concreteCompressiveStrength: num [1:1030] 80 61.9 40.3 41.1 44.3 ...
>
```

After that we looked at the summary of our data with summary() function. This function gives us minimum, maximum, mean , median values for each variable, and outputs if the column has NA value. In our case no column has NA values, so no need to apply na.omit function.

```
> # To get the statistical summary of our data
> summary(df)
     cement      blastFurnaceSlag     flyAsh           water       superplasticizer coarseAggregate
 Min.   :102.0   Min.   :  0.0    Min.   :  0.00   Min.   :121.8   Min.   : 0.000   Min.   : 801.0
 1st Qu.:192.4   1st Qu.:  0.0    1st Qu.:  0.00   1st Qu.:164.9   1st Qu.: 0.000   1st Qu.: 932.0
 Median :272.9   Median : 22.0    Median :  0.00   Median :185.0   Median : 6.350   Median : 968.0
 Mean   :281.2   Mean   : 73.9    Mean   : 54.19   Mean   :181.6   Mean   : 6.203   Mean   : 972.9
 3rd Qu.:350.0   3rd Qu.:142.9    3rd Qu.:118.27   3rd Qu.:192.0   3rd Qu.:10.160   3rd Qu.:1029.4
 Max.   :540.0   Max.   :359.4    Max.   :200.10   Max.   :247.0   Max.   :32.200   Max.   :1145.0
 fineAggregate        age         concreteCompressiveStrength
 Min.   :594.0   Min.   :  1.00   Min.   : 2.332
 1st Qu.:731.0   1st Qu.:  7.00   1st Qu.:23.707
 Median :779.5   Median : 28.00   Median :34.443
 Mean   :773.6   Mean   : 45.66   Mean   :35.818
 3rd Qu.:824.0   3rd Qu.: 56.00   3rd Qu.:46.136
 Max.   :992.6   Max.   :365.00   Max.   :82.599
```

Another statistical summary by psych package's describe() function. This function shows most values from summary function, but also adds another statistical properties as skew, standard deviation , kurtosis, etc.

```
> # Another statistical summary by psych package
> psych::describe(df)
                           vars    n   mean     sd median trimmed    mad    min    max  range  skew kurtosis   se
cement                        1 1030 281.17 104.51 272.90  273.47 117.72 102.00  540.0 438.00  0.51    -0.53 3.26
blastFurnaceSlag              2 1030  73.90  86.28  22.00   62.43  32.62   0.00  359.4 359.40  0.80    -0.52 2.69
flyAsh                        3 1030  54.19  64.00   0.00   46.85   0.00   0.00  200.1 200.10  0.54    -1.33 1.99
water                         4 1030 181.57  21.36 185.00  181.19  19.27 121.75  247.0 125.25  0.07     0.11 0.67
superplasticizer              5 1030   6.20   5.97   6.35    5.56   7.87   0.00   32.2  32.20  0.91     1.39 0.19
coarseAggregate               6 1030 972.92  77.75 968.00  973.49  68.64 801.00 1145.0 344.00 -0.04    -0.61 2.42
fineAggregate                 7 1030 773.58  80.18 779.51  776.41  67.44 594.00  992.6 398.60 -0.25    -0.11 2.50
age                           8 1030  45.66  63.17  28.00   32.53  31.13   1.00  365.0 364.00  3.26    12.07 1.97
concreteCompressiveStrength   9 1030  35.82  16.71  34.44   34.96  16.20   2.33   82.6  80.27  0.42    -0.32 0.52
```

The next step is scaling of the data. The main purpose of scaling is to normalize so that larger values for certain variables don't overwhelm the smaller values of other variables. There are two ways of normalization. The first is min-max normalization which linearly tranforms x to y with formula y = (x-min)/(max-min) where min and max are minimum and maximum values in that variable. The second way of normalization is called z-score normalization which tranforms input set into output set whose mean is almost 0 and standard deviation is close to 1. Let's apply both normalization techniques to our data:

Before normalization:

```
> print(df[1:6, ])
# A tibble: 6 × 9
  cement blastFurnaceSlag flyAsh water superplasticizer coarseAggregate fineAggregate   age concreteCompressiveStrength
   <dbl>            <dbl>  <dbl> <dbl>            <dbl>           <dbl>         <dbl> <dbl>                     <dbl>
1   540                0      0   162              2.5            1040           676    28                      80.0
2   540                0      0   162              2.5            1055           676    28                      61.9
3   332.             142.     0   228              0               932           594   270                      40.3
4   332.             142.     0   228              0               932           594   365                      41.1
5   199.             132.     0   192              0               978.          826.  360                      44.3
6   266              114      0   228              0               932           670    90                      47.0
```

Min-max normalization:

```
> print(df.minMaxNormalized[1:6, ])
     cement blastFurnaceSlag flyAsh     water superplasticizer coarseAggregate fineAggregate       age concreteCompressiveStrength
1 1.0000000        0.0000000      0 0.3213573       0.07763975       0.6947674     0.2057200 0.07417582                 0.9674449
2 1.0000000        0.0000000      0 0.3213573       0.07763975       0.7383721     0.2057200 0.07417582                 0.7419643
3 0.5262557        0.3964942      0 0.8483034       0.00000000       0.3808140     0.0000000 0.73901099                 0.4726417
4 0.5262557        0.3964942      0 0.8483034       0.00000000       0.3808140     0.0000000 1.00000000                 0.4823996
5 0.2205479        0.3683918      0 0.5608782       0.00000000       0.5156977     0.5807827 0.98626374                 0.5228058
6 0.3744292        0.3171953      0 0.8483034       0.00000000       0.3808140     0.1906673 0.24450549                 0.5568641
>
```

Z-score normalization:

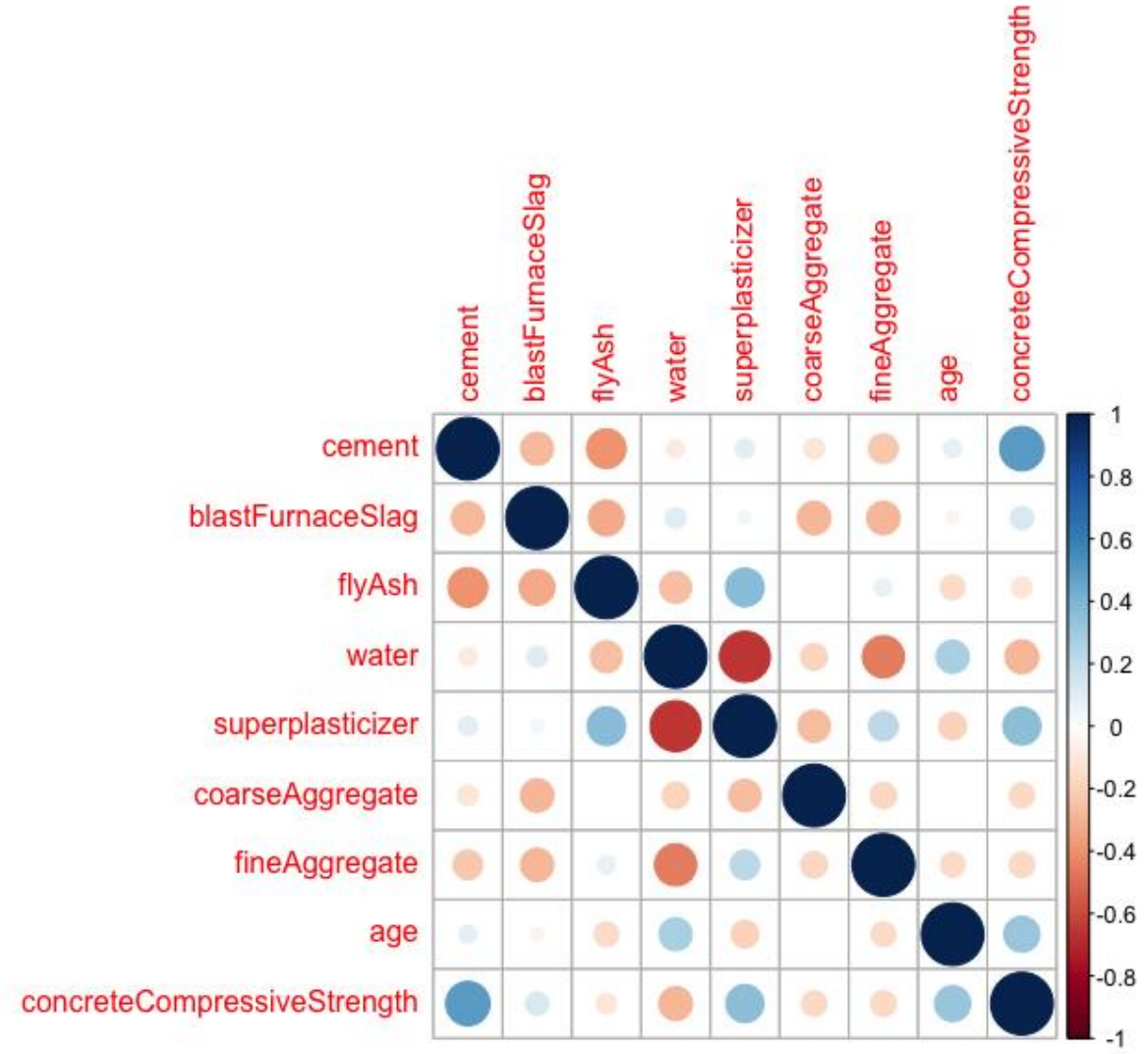
```
> print(df.zScoreNormalized[1:6, ])
      cement blastFurnaceSlag     flyAsh      water superplasticizer coarseAggregate fineAggregate        age concreteCompressiveStrength
1  2.4767147       -0.8564702 -0.8467207 -0.9162182       -0.6199241      0.86274101    -1.2170672 -0.2795973                 2.6439078
2  2.4767147       -0.8564702 -0.8467207 -0.9162182       -0.6199241      1.05565758    -1.2170672 -0.2795973                 1.5605190
3  0.4912044        0.7951464 -0.8467207  2.1743108       -1.0384398     -0.52625830    -2.2398245  3.5513405                 0.2664782
4  0.4912044        0.7951464 -0.8467207  2.1743108       -1.0384398     -0.52625830    -2.2398245  5.0552210                 0.3133631
5 -0.7900477        0.6780844 -0.8467207  0.4885677       -1.0384398      0.07049696     0.6475939  4.9760694                 0.5075064
6 -0.1451157        0.4648230 -0.8467207  2.1743108       -1.0384398     -0.52625830    -1.2919031  0.7018826                 0.6711497
```

As the difference of these two normalization techniques, we see that z-score normalization contains negative values as well, while min-max contains values from 0 to 1.

To pick the most correlated attributes for the further analysis we decided to closely examine correlations in our dataset. With cor() function we got the correlation matrix of our attributes:

```
> print(df.correlation)
                              cement blastFurnaceSlag       flyAsh       water superplasticizer coarseAggregate fineAggregate         age concreteCompressiveStrength
cement                    1.00000000      -0.27519344 -0.397475440 -0.08154361      0.09277137    -0.109356039   -0.22272017  0.081947264                0.4978327
blastFurnaceSlag         -0.27519344       1.00000000 -0.323569468  0.10728594      0.04337574    -0.283998230   -0.28159326 -0.044245801                0.1348244
flyAsh                   -0.39747544      -0.32356947  1.000000000 -0.25704400      0.37733956    -0.009976788    0.07907635 -0.154370165                -0.1057533
water                    -0.08154361       0.10728594 -0.257043997  1.00000000     -0.65746444    -0.182311668   -0.45063498  0.277604429                -0.2896135
superplasticizer          0.09277137       0.04337574  0.377339559 -0.65746444      1.00000000    -0.266302755    0.22250149 -0.192716518                0.3661023
coarseAggregate          -0.10935604      -0.28399823 -0.009976788 -0.18231167     -0.26630276     1.000000000   -0.17850575 -0.003015507                -0.1649278
fineAggregate            -0.22272017      -0.28159326  0.079076351 -0.45063498      0.22250149    -0.178505755    1.00000000 -0.156094049                -0.1672490
age                       0.08194726      -0.04424580 -0.154370165  0.27760443     -0.19271652    -0.003015507   -0.15609405  1.000000000                0.3288770
concreteCompressiveStrength 0.49783272     0.13482445 -0.105753348 -0.28961348      0.36610230    -0.164927821   -0.16724896  0.328876976                1.0000000
```

In order to better visualize it we used corrplot package's corrplot() function:



Looking at correlation matrix and plot, Blast Furnace Slag seems to be very little correlation with most of other attributes, therefore it can be removed. Another attribute that is not also so correlated with others is age, again it has very little correlation, so it will be removed to decrease dimensionality of analysis. Lastly, we decided to remove fly ash attribute because majority of the values of fly ash is 0, even the median of it is 0, we thought that it will negatively affect further analysis, therefore decided to remove it.

Part 3

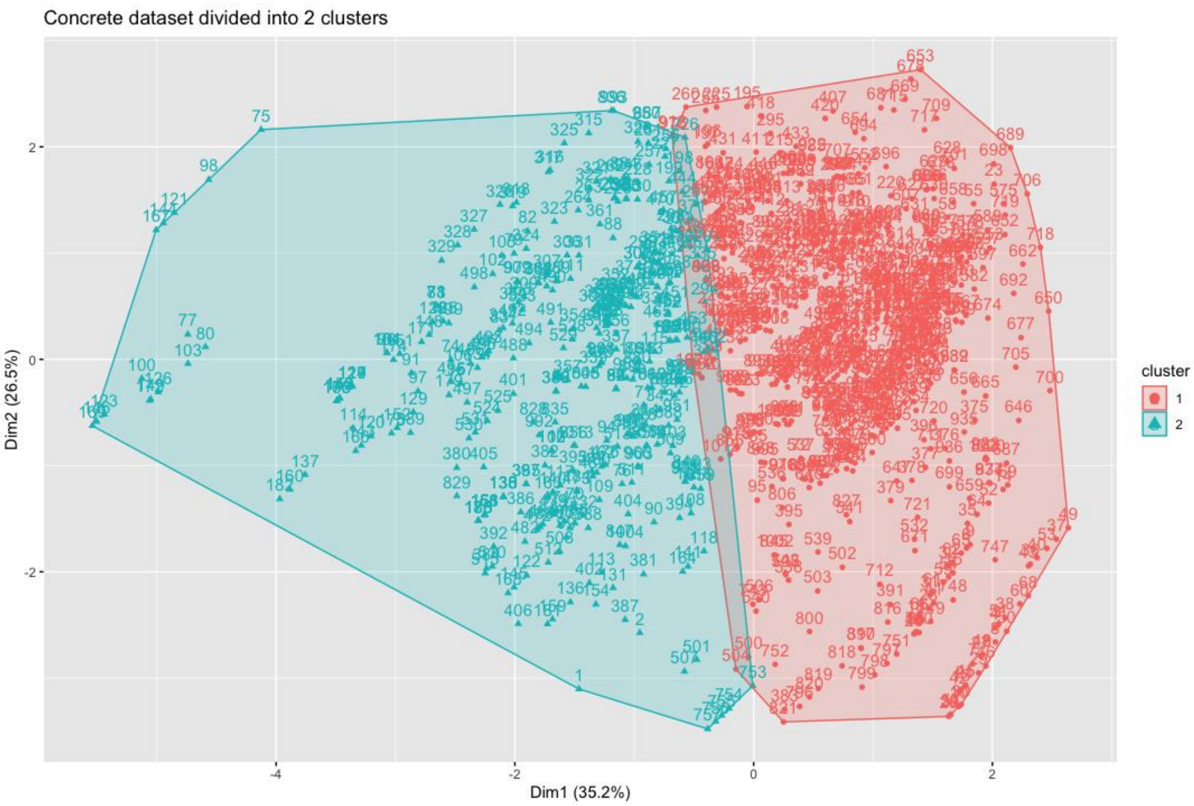
Application of K-means to our data.

With k = 2:

```
> df.simplified.scaled.k2 <- kmeans(df.simplified.scaled, centers=2, nstart=30)
> print(df.simplified.scaled.k2)
K-means clustering with 2 clusters of sizes 675, 355

Cluster means:
      cement      water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1 -0.2414237  0.4885580       -0.4826597       0.1267122    -0.1367778                  -0.4355053
2  0.4590451 -0.9289483        0.9177332      -0.2409316     0.2600704                   0.8280734

Clustering vector:
   [1] 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
  [65] 1 1 1 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [129] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
 [193] 1 1 1 1 1 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2
 [257] 2 2 2 1 2 2 2 2 1 1 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2
 [321] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 1 2 2 2 2 2 1 1 1 1 1 2 2 2 1 2
 [385] 2 2 2 2 2 1 1 2 2 2 1 1 2 1 1 1 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 1 1 1 2 2 1 1 2 1
 [449] 1 1 2 2 2 1 1 2 2 1 1 2 1 2 1 2 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2 1 1 1 1 1 2 2 2 2 2 2
 [513] 2 2 2 1 1 1 1 2 1 1 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [577] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [641] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [705] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1
 [769] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 1 2 1
 [833] 1 2 2 2 1 1 1 1 2 1 1 2 1 2 2 2 1 1 1 1 1 1 1 2 2 1 2 1 2 2 1 2 1 1 1 2 2 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 1 2
 [897] 2 2 1 1 1 1 2 1 1 1 2 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 2 1 1 2 2 1 1 1 1 1 1 1 2 2
 [961] 2 1 1 1 1 2 1 1 1 2 2 2 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 2 2 2 1 1 1 1 2 1 1
 [ reached getOption("max.print") -- omitted 30 entries ]

Within cluster sum of squares by cluster:
[1] 2936.834 1759.782
 (between_SS / total_SS =  23.9 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

between_ss/total_ss = 23.9 %

The visualization of clusters:



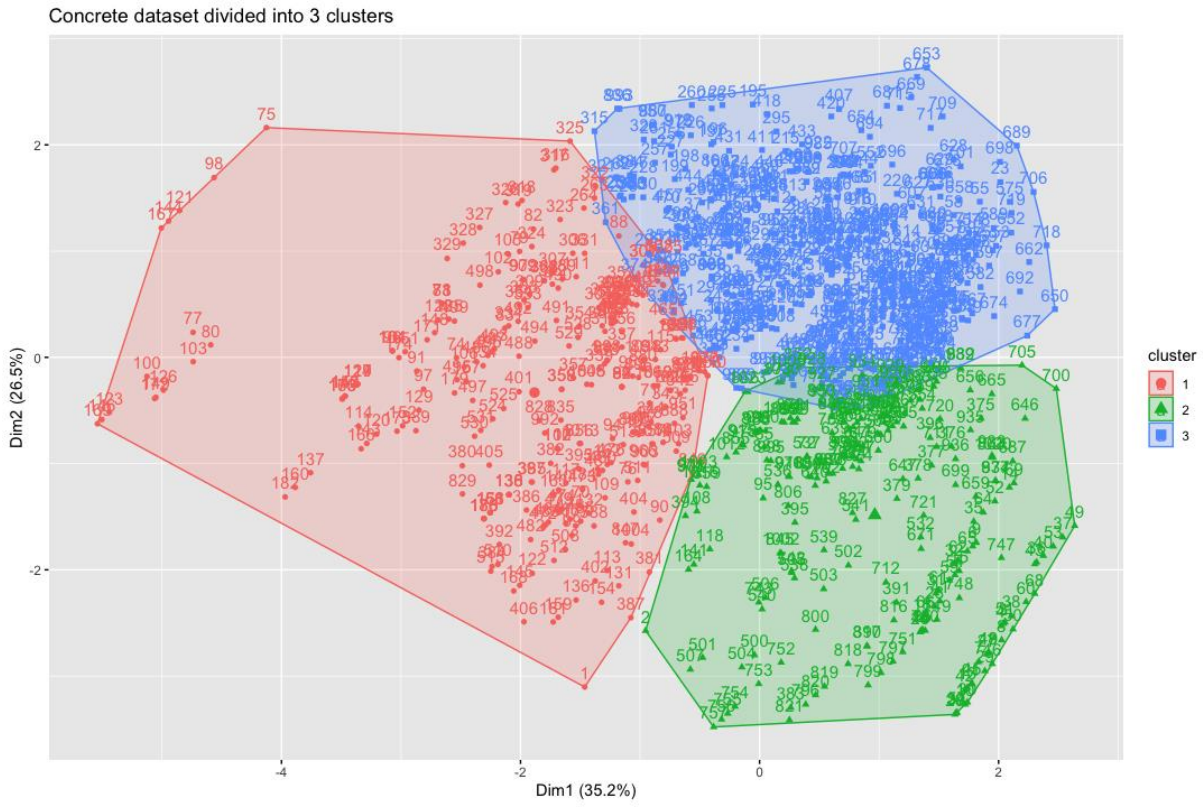
As we see two clusters are obviously separated with small overlapping area.

With k = 3:

```
> print(df.simplified.scaled.k3)
K-means clustering with 3 clusters of sizes 280, 233, 517

Cluster means:
      cement       water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1  0.6715274 -0.93609090        1.0388568      -0.4959676     0.3051519                  0.9627386
2  0.5706330  1.02116045       -0.6308882      -0.1651980    -1.1648801                  0.2061103
3 -0.6208610  0.04676029       -0.2783036       0.3430601     0.3597186                 -0.6142950

Clustering vector:
   [1] 1 2 2 2 3 2 2 2 2 2 3 3 2 2 2 2 3 2 2 2 2 3 3 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 3 3 2 3 2 2 2 2 3 2
  [65] 2 2 3 2 2 1 1 2 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 3 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1
 [129] 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3
 [193] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [257] 3 3 3 3 3 3 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [321] 3 1 1 1 1 1 1 1 3 1 1 1 1 3 3 1 1 1 3 3 1 1 1 3 3 3 1 1 3 3 1 1 1 3 3 1 1 1 3 3 1 1 1 3 3 1 1 1 3 3 1 1 1 3 3 3 3 3 2 2 2 2 2 1 1 1 2 1
 [385] 1 1 1 1 1 2 2 1 1 2 2 2 1 3 3 3 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3
 [449] 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 1 3 3 3 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 1 1 1 1 1 1
 [513] 1 1 1 3 3 3 3 1 3 3 1 1 1 1 1 1 1 1 2 3 3 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 2 3 2 2 3 3 3 3 2 3 3 2 3 2 3 2 3 3
 [577] 3 3 3 3 2 3 3 2 2 2 3 3 3 3 3 3 2 3 2 3 3 2 3 3 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2 2
 [641] 2 3 3 3 3 2 2 3 3 3 3 3 3 3 3 2 3 3 2 3 3 3 3 3 2 3 3 3 2 3 3 2 3 2 2 3 3 3 3 3 3 3 3 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 2 2 3 3
 [705] 2 3 3 3 3 3 2 2 3 3 3 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 2 2
 [769] 3 3 3 3 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [833] 3 1 1 3 2 2 3 3 1 2 3 1 2 1 1 1 3 3 3 2 1 2 2 1 1 3 2 3 1 1 2 1 2 2 3 1 1 1 3 3 2 2 2 3 3 3 2 3 3 1 3 3 3 3 2 3 3 1 3 2 3 3 3 3 3 1 1
 [897] 1 1 3 3 3 3 1 3 3 3 1 2 1 3 3 3 2 1 2 3 2 3 3 3 3 3 3 2 3 2 2 2 2 3 2 3 2 2 2 2 2 2 3 3 2 3 1 3 3 3 3 2 3 3 1 3 2 3 3 3 3 3 1 1
 [961] 1 3 3 3 3 1 3 3 3 1 2 1 3 3 3 2 1 2 3 2 3 3 3 3 3 3 2 1 2 3 1 1 3 2 2 3 3 1 2 3
 [ reached getOption("max.print") -- omitted 30 entries ]

Within cluster sum of squares by cluster:
[1] 1234.614  987.737 1616.077
 (between_SS / total_SS =  37.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

between_ss/total_ss has improved by about 14% to 37,8%.

The visualization of clusters:



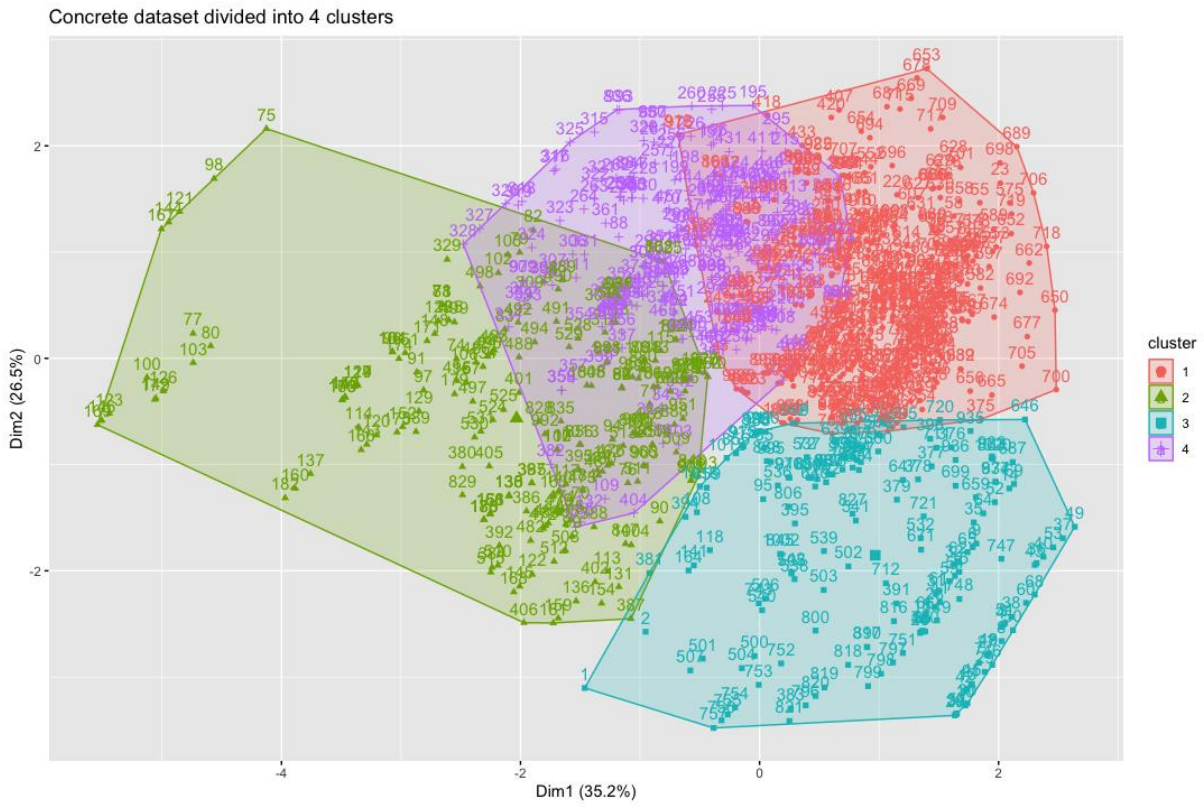
We can clearly observe three clusters on the visual, with some overlapping areas between red and blue clusters, as well as between blue and green clusters.

With k = 4:

```
> print(df.simplified.scaled.k4)
K-means clustering with 4 clusters of sizes 414, 212, 175, 229

Cluster means:
      cement      water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1 -0.4438936  0.4623507       -0.5902661      0.06033513     0.2702929                -0.72966656
2  0.9346236 -0.8368544        1.0963298     -0.89509173     0.3239363                 1.02657697
3  0.7840730  1.0847321       -0.6972754     -0.01648820    -1.3861650                 0.43108191
4 -0.6619259 -0.8900794        0.5850282      0.73216654     0.2707559                 0.03933756

Clustering vector:
   [1] 3 3 3 3 1 3 3 3 3 3 1 1 3 3 3 3 1 3 3 3 3 1 1 1 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 1 1 3 1 3 3 3 3 1 3
  [65] 3 3 1 3 3 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 3 4 2 4 2 2 2 2 1 2 3 2 2 2 2 2 2 2 2 2 2 2 3 4 2 4 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2
 [129] 2 2 2 4 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 4 2 2 2 2 2 2 2 1 1 1 1 1 1 1
 [193] 1 1 4 4 4 4 4 4 4 4 4 4 4 1 1 1 4 4 1 1 1 1 1 4 4 4 4 4 1 4 1 4 4 4 4 4 4 4 1 4 4 4 4 1 1 4 4 4 1 1 1 1 1 1 1 1 1 1 1 1 1 4 4
 [257] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 2 2 2 1 4 4 2 2 4 4 4 4 4 4
 [321] 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 3 3 3 3 2 3 4 3 2
 [385] 2 2 2 2 2 3 3 2 2 3 3 3 2 1 1 1 2 2 4 4 2 2 1 4 1 1 4 4 4 4 4 1 4 1 1 1 4 1 4 4 4 4 4 4 1 4 4 1 1 4 1 4 4 4 4 4 4 1 4 4 1 4 4 4
 [449] 4 4 4 4 4 4 1 4 4 1 4 4 4 4 4 4 4 4 4 1 4 4 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 2 2 2 2
 [513] 2 2 2 1 1 1 1 2 1 1 2 2 2 2 2 2 2 2 2 3 1 1 1 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 1 3 1 1 1 1 1 1 3 1 3 1 1
 [577] 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3
 [641] 3 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 3 1 1 1 1 1 1 3 1 1 1 1 1 3 1 3 1 1 1 1 1 1 1 1 1 3 1 1 3 1 1
 [705] 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 1 1 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1
 [769] 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 1 1 1 1 3 3 1 1 1 1 1 3 3 1 1 1 1 1 1 1 3 3 3 3 3 1 1 1 1 1 3 3 2 2 3 2 1
 [833] 1 4 2 4 3 3 4 1 2 3 4 4 3 2 2 2 1 4 4 1 2 3 3 2 2 1 3 1 2 2 3 2 1 1 4 2 2 2 1 4 1 3 3 1 1 1 1 2 1 4 1 4 1 1 4 2 1 1 1 4 1 1 1 2
 [897] 2 4 1 1 4 1 2 1 1 1 2 2 4 1 1 1 3 4 3 1 3 1 1 1 4 1 1 3 1 1 1 1 1 1 1 1 3 1 3 3 3 3 1 1 1 1 2 1 4 1 4 1 1 4 2 1 1 1 4 1 1 1 2 2
 [961] 4 1 1 4 1 2 1 1 1 2 2 4 1 1 1 3 4 3 1 3 1 1 1 4 1 1 3 2 1 1 4 2 4 3 3 4 1 2 3 4
 [ reached getOption("max.print") -- omitted 30 entries ]

Within cluster sum of squares by cluster:
[1] 1172.2659  839.8498  750.4029  573.5662
 (between_SS / total_SS =  46.0 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

between_ss/total_ss has improved by about 8% to 46,0%.

The visualization of clusters:



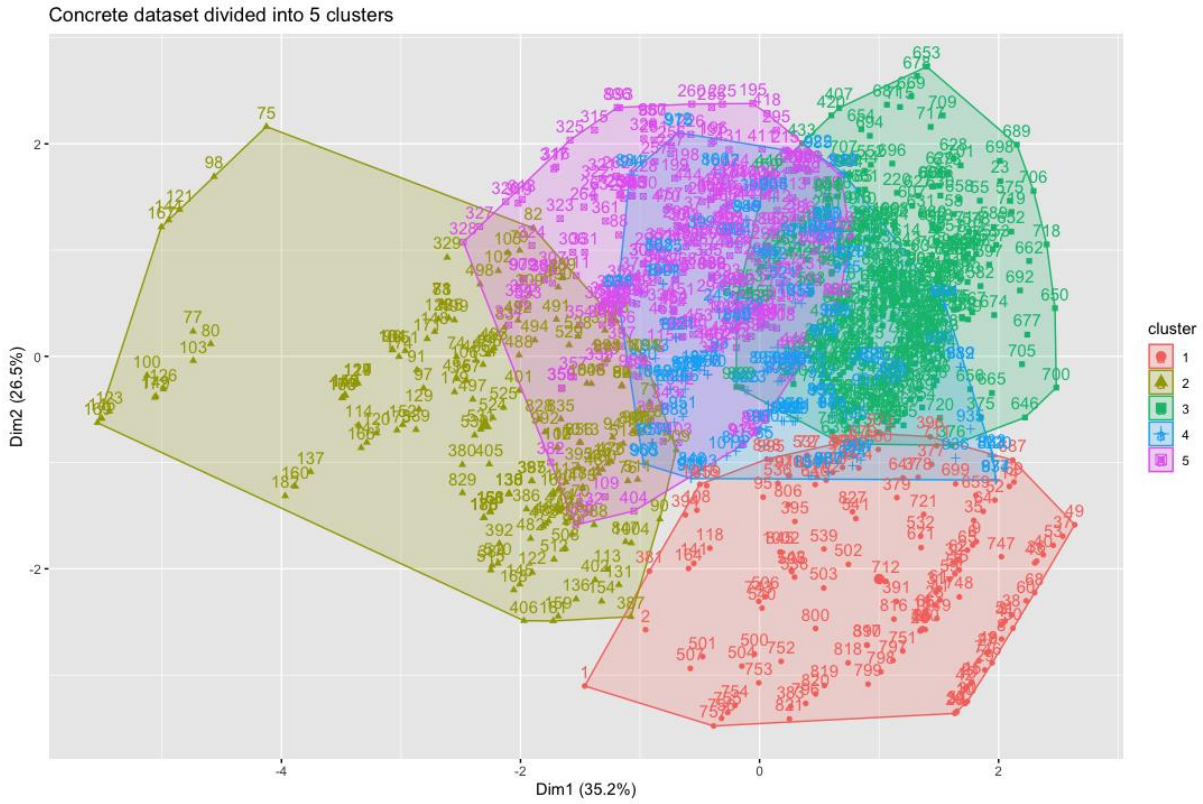
Now the overlapping area has significantly increased with purple cluster.

With k = 5:

```
> print(df.simplified.scaled.k5)
K-means clustering with 5 clusters of sizes 141, 174, 340, 131, 244

Cluster means:
      cement      water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1  0.9938882  1.1014595       -0.8376235       0.1067999    -1.4846383                0.551606307
2  1.1105967 -1.0111274        1.2087008      -0.8123336     0.3556636                1.176064706
3 -0.3280675  0.4441503       -0.8343346       0.3322632     0.3112836               -0.751146954
4 -0.5668700  0.5957032        0.3748525      -1.2372398    -0.1439414               -0.200420682
5 -0.6048325 -0.8541716        0.5834388       0.7188368     0.2478214               -0.003142108

Clustering vector:
   [1] 1 1 1 1 3 1 1 1 1 1 3 3 1 1 1 1 3 1 1 1 1 3 3 3 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 1 1 1 1 1 3 3 1 3 1 1 1 1 3 1
  [65] 1 1 3 1 1 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 4 5 2 5 2 2 2 2 4 2 1 2 2 2 2 2 2 2 2 2 2 1 5 2 5 2 2 2 5 4 2 1 2 2 2 2 2 2 2 2 2
 [129] 2 2 2 5 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 5 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 5 2 2 2 2 2 2 2 3 3 3 3 3 3 3
 [193] 3 3 5 5 5 5 5 5 5 5 5 5 3 3 3 5 5 3 3 3 3 3 5 5 5 5 3 5 5 3 5 5 5 5 5 5 5 3 5 5 5 5 3 3 3 3 3 3 3 3 4 4 3 3 3 3 3 5 5
 [257] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 3 5 3 3 3 3 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 2 5 5 5 2 2 5 5 5 5 5
 [321] 5 5 5 5 5 5 5 5 2 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 3 3 1 1 1 2 1 5 1 2
 [385] 2 2 2 2 2 1 1 2 2 1 1 1 2 4 4 3 2 2 5 5 2 2 3 5 3 5 5 5 5 5 5 3 5 5 3 3 5 3 5 5 5 5 5 3 5 5 3 3 5 3 5 5 5 5 5 3 5 5 3 3 5 5
 [449] 5 5 5 5 5 5 3 5 5 3 3 5 5 5 5 5 5 5 3 5 5 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 2 2 2 2
 [513] 2 2 2 3 3 3 3 4 4 4 4 2 2 5 5 2 2 2 2 1 3 3 3 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 3 3 3 3 3 3 3 3 1 3 1 3 3
 [577] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1
 [641] 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 4 3 3 1 3 3 3 3 3 3 3 1 3 3 3 3 3 4 3 1 3 3 3 3 3 3 3 3 3 1 3 3 1 3 3
 [705] 3 3 3 3 3 3 1 1 3 3 3 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 3 3 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3
 [769] 3 3 3 3 3 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 1 1 1 1 3 3 3 3 1 1 3 1 3 3 3 3 3 3 3 1 1 1 1 1 1 3 3 3 3 1 1 2 2 4 4 4
 [833] 4 5 2 5 4 1 5 3 2 4 5 4 1 4 2 2 4 5 5 4 4 4 4 2 4 4 1 4 2 4 4 4 4 4 5 4 4 4 4 5 4 4 5 4 4 5 3 4 4 3 4 3 5 4 4 4 4 5 4 5 4 4 5 4 4 4 5
 [897] 2 5 3 5 5 4 4 3 4 5 4 4 5 3 4 4 1 5 5 4 4 3 4 5 4 4 4 4 4 4 4 4 4 4 4 4 3 4 4 4 4 4 5 3 4 4 3 4 3 5 4 4 4 4 5 4 5 4 4 5 4 4 4 5 2
 [961] 5 3 5 5 4 4 3 4 5 4 4 5 3 4 4 1 5 5 4 4 3 4 5 4 4 4 4 4 4 4 5 2 5 4 1 5 3 2 4 5
 [ reached getOption("max.print") -- omitted 30 entries ]

Within cluster sum of squares by cluster:
[1] 580.7694 675.0542 766.9409 347.8788 605.0089
 (between_SS / total_SS =  51.8 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

between_ss/total_ss has improved by about 5% to 51,8%.

The visualization of clusters:



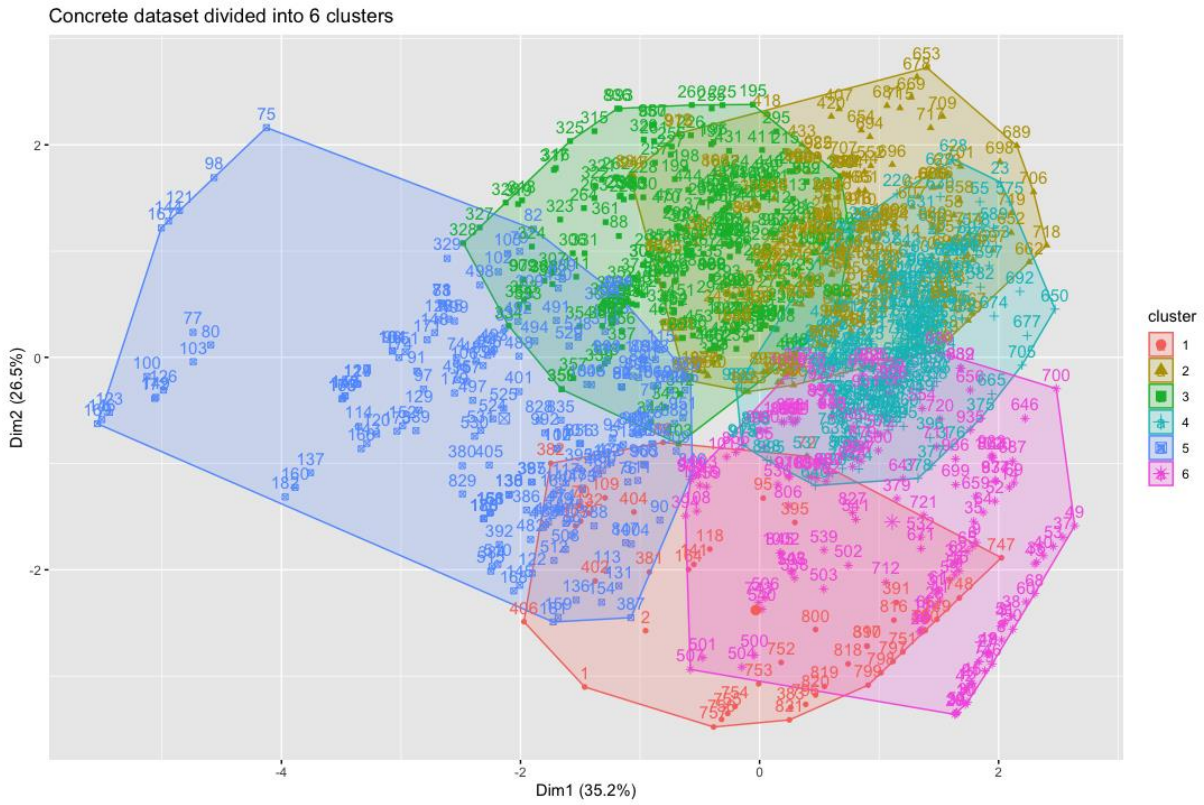
Again, purple and blue clusters are significantly overlapping with each other and with other clusters.

With k = 6:

```
> print(df.simplified.scaled.k6)
K-means clustering with 6 clusters of sizes 44, 187, 225, 234, 195, 145

Cluster means:
       cement      water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1  1.760191197 -0.2212749       -0.3960943       1.7033311   -1.81535752                1.058079887
2 -0.797677646  0.4584287       -0.3937869      -0.6356196    0.77496165               -0.718500527
3 -0.695457288 -0.8724555        0.5603348       0.6980183    0.33166001               -0.008926598
4 -0.003675685  0.3910572       -0.8101234       0.7061433   -0.06469834               -0.636826205
5  0.985521260 -0.8916501        1.1673568      -0.8973427    0.34150355                1.058313576
6  0.254335596  1.3977710       -0.5039635      -0.7130725   -1.31806470                0.223853469

Clustering vector:
   [1] 1 1 6 6 2 6 6 6 6 6 2 2 6 6 6 6 4 6 6 6 6 4 4 4 6 6 6 6 6 6 6 6 2 6 6 6 6 6 6 6 6 6 6 6 6 4 6 6 6 6 6 6 6 4 2 6 2 6 6 6 6 4 6
  [65] 6 6 4 6 6 1 5 1 5 5 5 5 5 5 5 5 5 5 5 5 6 1 5 3 5 5 5 5 3 5 1 5 5 5 5 5 5 5 5 5 5 6 1 5 3 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5
 [129] 5 5 5 1 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 1 5 5 5 5 5 5 5 5 2 2 2 2 2 2 2 2 2
 [193] 2 2 3 3 3 3 3 3 3 3 3 3 4 4 4 3 3 4 4 4 4 4 3 3 3 3 3 4 3 4 3 3 3 3 3 3 3 4 3 3 3 3 4 4 3 3 3 4 4 4 4 4 2 2 2 2 2 2 2 2 2 3 3
 [257] 3 3 3 3 3 3 3 3 3 3 3 3 4 3 3 3 3 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 5 3 3 3 3 5 2 3 3 5 5 3 3 3 3 3
 [321] 3 3 3 3 3 3 3 3 5 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 6 5 1 1 1 5
 [385] 5 5 5 5 5 1 1 5 5 6 1 4 5 2 2 4 5 1 3 1 5 1 2 3 4 3 3 3 3 3 3 2 3 2 4 2 3 4 3 3 3 3 3 3 2 3 3 4 2 3 4 3 3 3 3 3 3 2 3 3 4 3 3 3
 [449] 3 3 3 3 3 3 2 3 3 4 3 3 3 3 3 3 3 3 3 2 3 3 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 5 5 5 5 5
 [513] 5 5 5 2 2 2 2 2 2 2 2 5 5 5 5 5 5 5 5 6 2 2 4 6 4 6 6 6 6 6 2 2 2 2 2 2 2 2 2 4 2 4 6 4 4 4 4 4 6 4 4 4 4 4 4 4 4 4 6 4 4 4 4
 [577] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 4 4
 [641] 4 4 4 4 4 6 6 2 2 4 2 2 2 2 2 6 2 2 6 2 2 2 2 2 4 2 2 6 2 4 6 4 2 4 4 2 4 2 6 4 2 2 2 2 6 2 6 6 4 2 2 4 4 4 2 4 2 2 2 6 6 2 6 2 2
 [705] 4 2 2 2 2 2 6 6 2 2 2 2 2 2 2 6 6 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 6 6 4 4 4 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4
 [769] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1 4 4 4 4 6 6 4 4 4 4 4 4 4 4 4 1 1 1 1 1 4 4 4 4 4 6 5 5 6 5 6
 [833] 2 3 5 3 6 4 3 4 5 6 3 3 6 5 5 5 2 3 3 6 2 6 6 5 5 2 6 2 5 5 6 2 6 2 3 2 5 2 2 3 2 6 4 4 2 6 2 5 2 3 2 2 6 2 3 5 3 6 2 3 2 2 6 5
 [897] 5 3 4 2 3 2 5 4 2 2 5 6 3 2 2 2 6 3 4 2 6 2 2 3 3 2 2 6 2 6 6 6 6 6 6 4 6 6 6 6 4 4 2 6 2 5 2 3 2 2 6 2 3 5 3 6 2 3 2 2 6 5 5
 [961] 3 4 2 3 2 5 4 2 2 5 6 3 2 2 2 6 3 4 2 6 2 2 3 3 2 2 6 5 6 2 3 5 3 6 4 3 4 5 6 3
 [ reached getOption("max.print") -- omitted 30 entries ]

Within cluster sum of squares by cluster:
[1] 148.0157 421.7566 490.3450 416.0387 747.6665 453.4214
 (between_SS / total_SS =  56.6 %)

Available components:

[1] "cluster"      "centers"      "totss"        "withinss"     "tot.withinss" "betweenss"    "size"         "iter"
[9] "ifault"
```

between_ss/total_ss has improved by about 5% to 56,6%.

The visualization of clusters:

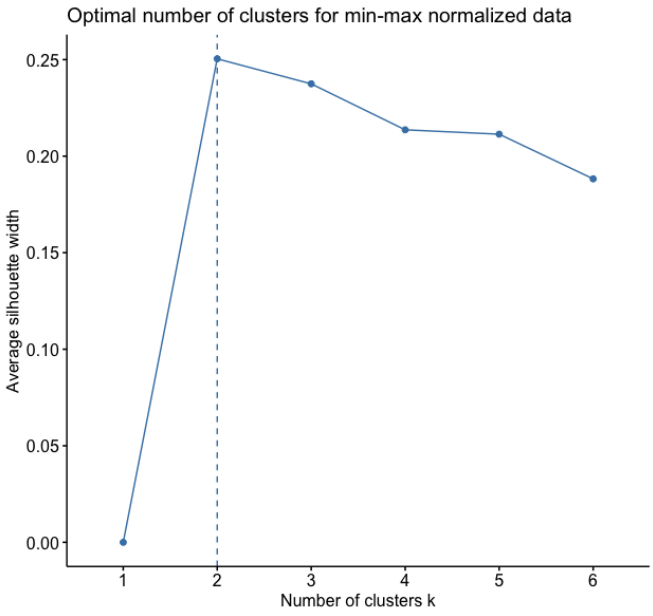


Here we can also observe a lot of overlapping areas between clusters.

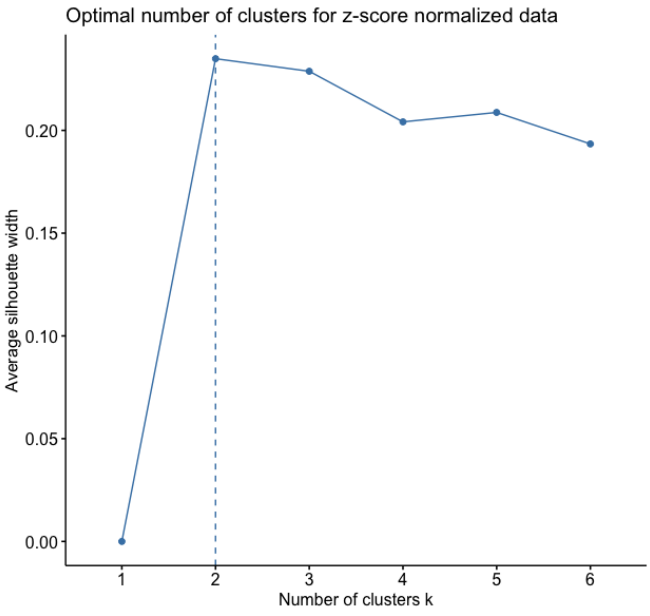
The same K-means analysis have been conducted with z score scaled data set as well. The results are very similar since the data set itself is the same. In order to keep the report file short, we decided not to include all screenshots of that, since the results are very similar to above presented results. The code is included in script file, as well result screenshots can be found in full resolution in img folder of the report.

The best number of clusters is obtained using fviz_nbclust() function from factoextra package:

Min-max normalized:



Z-score normalized:



As can be seen the optimal number of clusters in both cases is two clusters, and as for the graphics itself, both are very similar to each other.

Now let's analyze cluster centroids for optimal number of clusters (k=2):

```
> print(df.simplified.norm.k2$centers)
     cement     water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1 0.2747938 0.4980714        0.1470411       0.5533135     0.4869368                   0.3144444
2 0.6396697 0.4423708        0.2709730       0.4077815     0.3879786                   0.5936490
```

To analyze these numbers we need to look at the largest number in each column. Indicates that is most important cluster for that attribute. In case of min-max normalized data, the most important cluster for:
cement – 2, water – 1, superplasticizer - 2, coarseAggregate - 1, fineAggregate - 1, concreteCompressiveStrength - 2

```
> print(df.simplified.scaled.k2$centers)
      cement      water superplasticizer coarseAggregate fineAggregate concreteCompressiveStrength
1 -0.2414237  0.4885580       -0.4826597       0.1267122    -0.1367778                  -0.4355053
2  0.4590451 -0.9289483        0.9177332      -0.2409316     0.2600704                   0.8280734
```

In case of z-scored normalized data we can have negative values as well, since z-score allows negative values, and the most important cluster for:
cement – 2, water – 1, superplasticizer - 2, coarseAggregate - 1, fineAggregate - 2, concreteCompressiveStrength - 2

Comparing two normalization techniques the only difference is in fineAggregate, all other are the same for both techniques.

K-Nearest Neighbour

70/30 Ratio:

After dividing our data to train set and test set in the given ratio, KNN classification method is used with the following code where we first get kmeans of train set, then apply knn to test set and then comparing results with cross table, and finally computing accuracy.

```
# Evaluation of setA (70/30)

# Getting train set kmeans clusters according to our
# optimal number of k
df.simplified.norm.setA.train.k2 <- kmeans(df.simplified.norm.setA.train, centers = 2, nstart = 30)

# applying knn to test set
df.simplified.norm.setA.test.k2 <- knn(
  df.simplified.norm.setA.train,
  df.simplified.norm.setA.test,
  df.simplified.norm.setA.train.k2$cluster,
  k=2
)

# Getting real kmeans cluster for test set
df.simplified.norm.setA.test.kmeans.k2 <- kmeans(df.simplified.norm.setA.test, centers = 2, nstart = 30)

# Renaming for better understanding in cross table
knnPredictedValues <- df.simplified.norm.setA.test.k2
realKmeansClusterValues <- df.simplified.norm.setA.test.kmeans.k2$cluster

# Creating cross table
df.simplified.norm.setA.ct <- CrossTable(
  realKmeansClusterValues,
  knnPredictedValues,
  prop.chisq=FALSE
)

#compute accuracy
df.simplified.norm.setA.comparison <- data.frame(
  row.names = names(df.simplified.norm.setA.test.kmeans.k2$cluster),
  Actual=df.simplified.norm.setA.test.kmeans.k2$cluster,
  Predicted=df.simplified.norm.setA.test.k2
)
df.simplified.norm.setA.comparison$Predicted <- as.numeric(df.simplified.norm.setA.comparison$Predicted)
df.simplified.norm.setA.acc <- mean(df.simplified.norm.setA.comparison$Actual==df.simplified.norm.setA.comparison$Predicted)
print(df.simplified.norm.setA.acc)
```

We get the following cross table:

```
Total Observations in Table:  309


                       | knnPredictedValues
realKmeansClusterValues |         1 |         2 | Row Total |
-----------------------|-----------|-----------|-----------|
                     1 |        78 |         6 |        84 |
                       |     0.929 |     0.071 |     0.272 |
                       |     0.729 |     0.030 |           |
                       |     0.252 |     0.019 |           |
-----------------------|-----------|-----------|-----------|
                     2 |        29 |       196 |       225 |
                       |     0.129 |     0.871 |     0.728 |
                       |     0.271 |     0.970 |           |
                       |     0.094 |     0.634 |           |
-----------------------|-----------|-----------|-----------|
          Column Total |       107 |       202 |       309 |
                       |     0.346 |     0.654 |           |
-----------------------|-----------|-----------|-----------|
```

We can calculate the accuracy:
(78 + 196)/309 = 89%

60/40 Ratio:

```
Total Observations in Table:  412


                        | knnPredictedValues
realKmeansClusterValues |         1 |         2 | Row Total |
------------------------|-----------|-----------|-----------|
                      1 |       244 |        23 |       267 |
                        |     0.914 |     0.086 |     0.648 |
                        |     0.961 |     0.146 |           |
                        |     0.592 |     0.056 |           |
------------------------|-----------|-----------|-----------|
                      2 |        10 |       135 |       145 |
                        |     0.069 |     0.931 |     0.352 |
                        |     0.039 |     0.854 |           |
                        |     0.024 |     0.328 |           |
------------------------|-----------|-----------|-----------|
           Column Total |       254 |       158 |       412 |
                        |     0.617 |     0.383 |           |
------------------------|-----------|-----------|-----------|
```

We can calculate the accuracy:
(244 + 135)/412 = 92%

50/50 Ratio:

```
Total Observations in Table:  515


                        | knnPredictedValues
realKmeansClusterValues |         1 |         2 | Row Total |
------------------------|-----------|-----------|-----------|
                      1 |       315 |        19 |       334 |
                        |     0.943 |     0.057 |     0.649 |
                        |     0.969 |     0.100 |           |
                        |     0.612 |     0.037 |           |
------------------------|-----------|-----------|-----------|
                      2 |        10 |       171 |       181 |
                        |     0.055 |     0.945 |     0.351 |
                        |     0.031 |     0.900 |           |
                        |     0.019 |     0.332 |           |
------------------------|-----------|-----------|-----------|
           Column Total |       325 |       190 |       515 |
                        |     0.631 |     0.369 |           |
------------------------|-----------|-----------|-----------|
```

We can calculate the accuracy:
(315 + 171)/515 = 94%

	70/30	60/40	50/50
Accuracy	89%	92%	94%

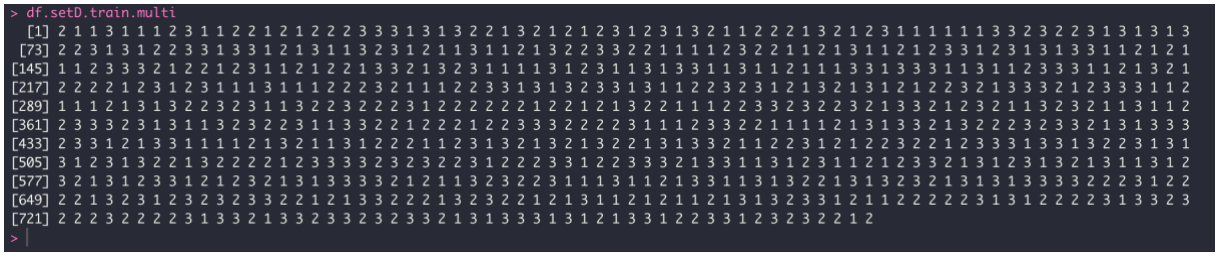
As can be seen, predictions are pretty good. It is an effect of removing the columns that are not so correlated and cleaning data properly.

Part 4

First of all, we specify values of Strength based on Concrete Compressive Strength. We used for that quantile() function. In Binomial Logistic Regression, values that fit to the first interval of the quartile got the value of 0 (which is associated with “low”) and 1 (= “high”) to the values of the second interval. For the Multinomial Logistic Regression values got the values depending on which interval they fit – 1, 2, 3 which are associated with “low”, “mid”, “high” respectively:

Binomial train data:

Multinomial train data:



To apply Binomial Logistic Regression, we used glm() function and fitted data into it, here is summary of it:

```
> df.bi.fit <- glm(strength ~
+                    cement + water + superplasticizer
+                  + coarseAggregate + fineAggregate,
+                  family = binomial, data = df.bi)
> summary(df.bi.fit)

Call:
glm(formula = strength ~ cement + water + superplasticizer +
    coarseAggregate + fineAggregate, family = binomial, data = df.bi)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.4468  -0.8743  -0.3318   0.9178   2.2120

Coefficients:
                Estimate Std. Error z value Pr(>|z|)
(Intercept)       2.1200     0.9883   2.145 0.031944 *
cement            3.9951     0.4458   8.961  < 2e-16 ***
water            -3.7098     0.9576  -3.874 0.000107 ***
superplasticizer  1.4300     0.8288   1.725 0.084442 .
coarseAggregate  -1.4239     0.5174  -2.752 0.005918 **
fineAggregate    -3.2200     0.5809  -5.543 2.97e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1070.09  on 771  degrees of freedom
Residual deviance:  828.41  on 766  degrees of freedom
AIC: 840.41

Number of Fisher Scoring iterations: 4
```

Here are accuracy, Confusion Matrix and Statistics of binomial logistic regression:

```
> accuracy <- mean(strength.predict.bi == df.setD.test.bi)
> accuracy
[1] 0.744186
> confusionMatrix(data=as.factor(strength.predict.bi), reference = as.factor(df.setD.test.bi))
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  92  34
         1  32 100

               Accuracy : 0.7442
                 95% CI : (0.6864, 0.7963)
    No Information Rate : 0.5194
    P-Value [Acc > NIR] : 1.092e-13

                  Kappa : 0.4879

 Mcnemar's Test P-Value : 0.902

            Sensitivity : 0.7419
            Specificity : 0.7463
         Pos Pred Value : 0.7302
         Neg Pred Value : 0.7576
             Prevalence : 0.4806
         Detection Rate : 0.3566
   Detection Prevalence : 0.4884
      Balanced Accuracy : 0.7441

       'Positive' Class : 0
```

If we take “high” / 1 as positive, and “low” / 0 as negative:

True Positive: 100
False Positive: 32
True Negative: 92
False Negative: 34

When applying Multinomial Logistic Regression, we used multinom() function:

```
> summary(df.multi.fit)
Call:
multinom(formula = strength ~ cement + water + superplasticizer +
    coarseAggregate + fineAggregate, data = df.multi)

Coefficients:
  (Intercept)   cement     water superplasticizer coarseAggregate fineAggregate
2    2.237312 2.788573 -1.721026         2.063894       -2.632908     -2.585437
3    3.866729 5.011216 -5.418774         2.416333       -3.039662     -4.564112

Std. Errors:
  (Intercept)    cement    water superplasticizer coarseAggregate fineAggregate
2    1.147545 0.5375038 1.121471        0.9344721       0.6142043     0.6534967
3    1.257288 0.5829728 1.230258        1.0337477       0.6730673     0.7450791

Residual Deviance: 1401.923
AIC: 1425.923
```

Here are accuracy, Confusion Matrix and Statistics of multinomial logistic regression:

```
> confusionMatrix(as.factor(strength.predicted.multi), as.factor(df.setD.test.multi))
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3
         1 57 26 11
         2 15 34 14
         3 13 23 65

Overall Statistics

               Accuracy : 0.6047
                 95% CI : (0.5421, 0.6647)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.4054

 Mcnemar's Test P-Value : 0.1506

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.6706   0.4096   0.7222
Specificity            0.7861   0.8343   0.7857
Pos Pred Value         0.6064   0.5397   0.6436
Neg Pred Value         0.8293   0.7487   0.8408
Prevalence             0.3295   0.3217   0.3488
Detection Rate         0.2209   0.1318   0.2519
Detection Prevalence   0.3643   0.2442   0.3915
Balanced Accuracy      0.7284   0.6220   0.7540
>
```

Next, we compared these results with our results that we got via KMeans algorithm with k = 3. Below, you can find the cross-tabulation of the results

As we can see there is: $\frac{(88+35+52)}{258} = 67.8\%$ similarity

```
> CrossTable(
+   realKmeansClusterValues,
+   prediction,
+   prop.chisq=FALSE
+ )


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  258


                        | prediction
realKmeansClusterValues |         1 |         2 |         3 | Row Total |
------------------------|-----------|-----------|-----------|-----------|
                      1 |        88 |        23 |         9 |       120 |
                        |     0.733 |     0.192 |     0.075 |     0.465 |
                        |     0.936 |     0.365 |     0.089 |           |
                        |     0.341 |     0.089 |     0.035 |           |
------------------------|-----------|-----------|-----------|-----------|
                      2 |         6 |        35 |        40 |        81 |
                        |     0.074 |     0.432 |     0.494 |     0.314 |
                        |     0.064 |     0.556 |     0.396 |           |
                        |     0.023 |     0.136 |     0.155 |           |
------------------------|-----------|-----------|-----------|-----------|
                      3 |         0 |         5 |        52 |        57 |
                        |     0.000 |     0.088 |     0.912 |     0.221 |
                        |     0.000 |     0.079 |     0.515 |           |
                        |     0.000 |     0.019 |     0.202 |           |
------------------------|-----------|-----------|-----------|-----------|
           Column Total |        94 |        63 |       101 |       258 |
                        |     0.364 |     0.244 |     0.391 |           |
------------------------|-----------|-----------|-----------|-----------|
```

Our next step was trying multinomial logistic regression with different set of attributes. We decided to remove Coarse Aggregate as according to our correlation matrix that we plotted in Part 2, it had the least correlation with other attributes from the remained ones. Here is the summary:

```
> summary(df.multi.fit2)
Call:
multinom(formula = strength ~ cement + water + superplasticizer +
    fineAggregate, data = df.multi)

Coefficients:
  (Intercept)   cement     water superplasticizer fineAggregate
2   -1.551117 2.855214  1.338801         4.138914     -1.508462
3   -0.499583 5.265651 -2.015841         4.809867     -3.316049

Std. Errors:
  (Intercept)    cement     water superplasticizer fineAggregate
2   0.7056402 0.5107448 0.8464800         0.7871917     0.5882505
3   0.7701214 0.5478341 0.9405725         0.8697434     0.6760458

Residual Deviance: 1427.787
AIC: 1447.787
```

You can find below the Confusion Matrix. As we can see that accuracy decreased this time, so the results were quite good for the first time when we used Multinomial Logistic Regression.

```
> confusionMatrix(as.factor(strength.predicted.multi2), as.factor(df.setD.test.multi))
Confusion Matrix and Statistics

          Reference
Prediction  1  2  3
         1 54 23 11
         2 18 37 16
         3 13 23 63

Overall Statistics

               Accuracy : 0.5969
                 95% CI : (0.5343, 0.6573)
    No Information Rate : 0.3488
    P-Value [Acc > NIR] : 4.278e-16

                  Kappa : 0.3941

 Mcnemar's Test P-Value : 0.5656

Statistics by Class:

                     Class: 1 Class: 2 Class: 3
Sensitivity            0.6353   0.4458   0.7000
Specificity            0.8035   0.8057   0.7857
Pos Pred Value         0.6136   0.5211   0.6364
Neg Pred Value         0.8176   0.7540   0.8302
Prevalence             0.3295   0.3217   0.3488
Detection Rate         0.2093   0.1434   0.2442
Detection Prevalence   0.3411   0.2752   0.3837
Balanced Accuracy      0.7194   0.6257   0.7429
```

We also compared new results with our results from K-means algorithm (k=3). This time similarity was slightly less: $\frac{83+36+50}{258} = 65.5\%$

```
> CrossTable(
+   realKmeansClusterValues,
+   prediction,
+   prop.chisq=FALSE
+ )


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  258


                         | prediction
realKmeansClusterValues |         1 |         2 |         3 | Row Total |
------------------------|-----------|-----------|-----------|-----------|
                      1 |        83 |        28 |         9 |       120 |
                        |     0.692 |     0.233 |     0.075 |     0.465 |
                        |     0.943 |     0.394 |     0.091 |           |
                        |     0.322 |     0.109 |     0.035 |           |
------------------------|-----------|-----------|-----------|-----------|
                      2 |         5 |        36 |        40 |        81 |
                        |     0.062 |     0.444 |     0.494 |     0.314 |
                        |     0.057 |     0.507 |     0.404 |           |
                        |     0.019 |     0.140 |     0.155 |           |
------------------------|-----------|-----------|-----------|-----------|
                      3 |         0 |         7 |        50 |        57 |
                        |     0.000 |     0.123 |     0.877 |     0.221 |
                        |     0.000 |     0.099 |     0.505 |           |
                        |     0.000 |     0.027 |     0.194 |           |
------------------------|-----------|-----------|-----------|-----------|
           Column Total |        88 |        71 |        99 |       258 |
                        |     0.341 |     0.275 |     0.384 |           |
------------------------|-----------|-----------|-----------|-----------|
```

Discussion

We found this project very helpful since it helped us to realize a few concepts and to transfer our theoretical knowledge into practical knowledge. Firstly, we learned the importance of plotting data to see the relationships between the attributes of the data which we did in the first part of our project. With the help of built-in functions and external package functions we could see the correlations of attributes in our data set, which helped us in the phases of our analysis.

Second important achievement was getting acquainted with Supervised and Unsupervised learning algorithms. In Part 3 we learned about clustering technique which is an unsupervised algorithm. We learned how to divide the given dataset into the clusters, and we also managed to learn how to find the most optimal number of clusters. We also learned about classification with K-Nearest Neighbor which is supervised algorithm. We implemented these two techniques in part 3, therefore we learned the ways of implementation of these techniques in R.

Lastly, we also learned about Linear and Logistic Regressions: Linear Regression while examining the tutorial posted on Blackboard and Logistic Regression during the Part 4 of this assignment. We learned about two ways to predict the values after training on the train dataset via Binomial and Multinomial Logistic Regressions.

To sum up, during this project we have learned several different techniques that are being used in data analysis and applied it in practice.