Pen-Based Recognition of Handwritten Digits Data Set- PCA Analysis

Diksha Srivastava (21235117)

13/02/2022

PCA

Principal component analysis helps in reducing the dimension of the multivariate data. It is important when the input data has multiple features. With the help of PCA, we can check if the first two principal components are enough to show the variance of the data.

Handwritten Digits Example

The dataset contains 10992 instances where each instance has 16 numeric features (f1, f2, f3,. . . , f16) representing the shape of the handwritten digits and 10 classes from 0 to 9. A classifier inputs these features and classifies it into one of the digits from 0 to 9. The below code is used to read the data from the CSV.

```
library(readr)
# Read the data
penDigits <- read_csv(file = "E:/Users/Diksha/Desktop/NUIG/DV/digits.csv")
head(penDigits)
```

```
## # A tibble: 6 x 17
##      f1    f2    f3    f4    f5    f6    f7    f8    f9   f10   f11   f12   f13
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1    88    92     2    99    16    66    94    37    70     0     0    24    42
## 2    80   100    18    98    60    66   100    29    42     0     0    23    42
## 3     0    94     9    57    20    19     7     0    20    36    70    68   100
## 4    95    82    71   100    27    77    77    73   100    80    93    42    56
## 5    68   100     6    88    47    75    87    82    85    56   100    29    75
## 6    70   100   100    97    70    81    45    65    30    49    20    33     0
## # ... with 4 more variables: f14 <dbl>, f15 <dbl>, f16 <dbl>, digits <dbl>
```

Feature Processing

The 'digits' column contains the numbers which vary between 0 to 9 and hence, is a set of unordered integer values. These integer values needs to be converted to factors, which is a data wrangling operation. Thus, we get the digits column from the dataset and convert it into factors from 0 to 9. We make use of dplyr's mutate function to add the column digits of type factor.

```
library(dplyr)
penDigits_factor <- penDigits %>% mutate(digits = case_when(
  digits == "0" ~ "0",
  digits == "1" ~ "1",
  digits == "2" ~ "2",
  digits == "3" ~ "3",
  digits == "4" ~ "4",
  digits == "5" ~ "5",
  digits == "6" ~ "6",
  digits == "7" ~ "7",
  digits == "8" ~ "8",
  digits == "9" ~ "9"
)) %>% mutate(digits = factor(digits, levels = c("0", "1", "2",
                                                 "3", "4",
                                                 "5","6","7",
                                                 "8","9")))
print(penDigits_factor)
```

```
## # A tibble: 10,992 x 17
##       f1    f2    f3    f4    f5    f6    f7    f8    f9   f10   f11   f12   f13
##    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1    88    92     2    99    16    66    94    37    70     0     0    24    42
##  2    80   100    18    98    60    66   100    29    42     0     0    23    42
##  3     0    94     9    57    20    19     7     0    20    36    70    68   100
##  4    95    82    71   100    27    77    77    73   100    80    93    42    56
##  5    68   100     6    88    47    75    87    82    85    56   100    29    75
##  6    70   100   100    97    70    81    45    65    30    49    20    33     0
##  7    40   100     0    81    15    58   100    57    47    87    50    88    40
##  8     3    71     0    95    45   100   100    99    79    78    48    53    31
##  9    79    87    98    81    71   100    72    73   100    66    91    21    48
## 10    92    95    30   100    34    68    87    89    84    78   100    35    64
## # ... with 10,982 more rows, and 4 more variables: f14 <dbl>, f15 <dbl>,
## #   f16 <dbl>, digits <fct>
```

Principal Component Analysis

In order to perform the PCA, we select the first 16 columns which represent the features in the dataset. We exclude the last column 'digits' which represents the class. We do not exclude any other column as they are important in the analysis of digits and there is a significant variance in each feature/column. We do not require any scaling as the values are in the range 0 to 100.

```
# Selecting the features to perform the PCA analysis on
library(FactoMineR)

pca <- select(penDigits_factor, -digits) %>% PCA(graph = FALSE)
print(pca)
```

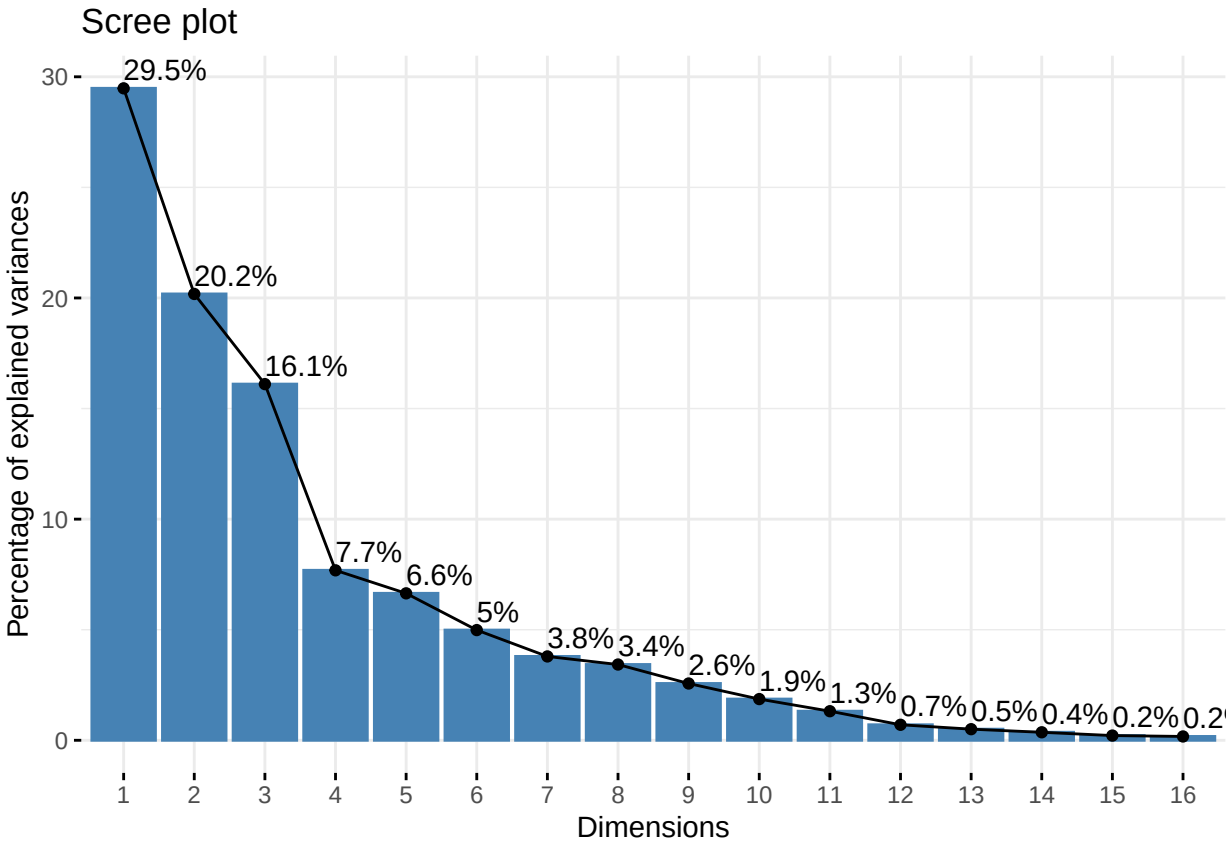
```
## **Results for the Principal Component Analysis (PCA)**
## The analysis was performed on 10992 individuals, described by 16 variables
## *The results are available in the following objects:
##
##    name               description
```

```
## 1  "$eig"             "eigenvalues"
## 2  "$var"             "results for the variables"
## 3  "$var$coord"       "coord. for the variables"
## 4  "$var$cor"         "correlations variables - dimensions"
## 5  "$var$cos2"        "cos2 for the variables"
## 6  "$var$contrib"     "contributions of the variables"
## 7  "$ind"             "results for the individuals"
## 8  "$ind$coord"       "coord. for the individuals"
## 9  "$ind$cos2"        "cos2 for the individuals"
## 10 "$ind$contrib"     "contributions of the individuals"
## 11 "$call"            "summary statistics"
## 12 "$call$centre"     "mean of the variables"
## 13 "$call$ecart.type" "standard error of the variables"
## 14 "$call$row.w"      "weights for the individuals"
## 15 "$call$col.w"      "weights for the variables"
```

Section 1- Scree Plot

The principal component generated represent the variance of the data. Generally, the first n components represent the variance of the data in a better way. In order to do 2-D plot, the first two components should be enough to represent the variance of the data. Thus, the scree plot helps in plotting a bar chart which represents the variance in percentage by the first n components in descending order.

```
# Using the scree plot to check the variance of components
library(factoextra)
pca_scree <- fviz_screeplot(pca, choice="variance", addlabels = TRUE, ncp = 16)
pca_scree
```



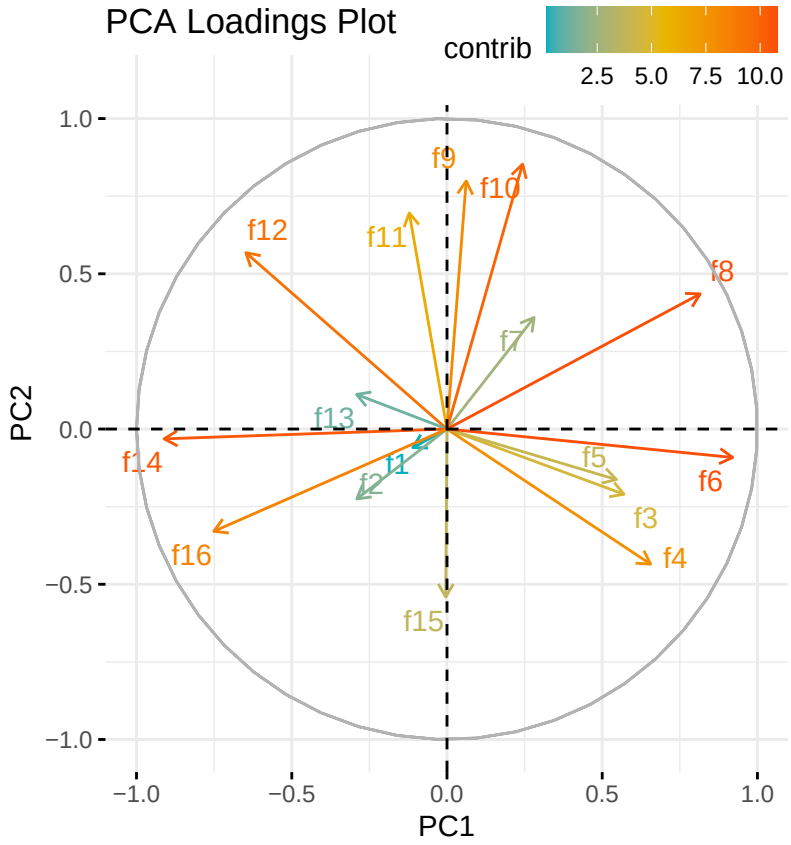
The scree plot shows that the first two principal components represent 49.7% of variance of the dataset. Since the percentage is large, hence, these components will be able to show good separation of data points in the scatter plot.

Loadings Plot

Loadings plot helps in identifying the relationship between the original variables and the principal components. The correlation with PC1 are plotted on the x-axis and with PC2 is plotted on y-axis. The variables are positively correlated if they are on the same side of the plot and negatively correlated if they are on opposite side of the plot. The distance from the origin conveys the impact of the variable on the model. We make use of 'fviz_pca_var' function to plot the same.

```
loadings_plot <- fviz_pca_var(pca,
                              col.var = "contrib",
                              gradient.cols = c("#00AFBB", "#E7B800", "#FC4E07"),
                              repel = TRUE) +
                  xlab("PC1") +
                  ylab("PC2") +
                  # Increasing the limit of y-axis scale.
                  ylim(c(-1,1.1))+
                  ggtitle("PCA Loadings Plot") +
                  # Changing the position of legend.
                  theme(
                        legend.box.background = element_rect(fill = "white", color = "white"),
                        legend.position = c(0.74, 1.01),
```

```
                      legend.direction = "horizontal",
                      legend.box.margin = margin(0.05, 0.05, 0.05, 0.05),
                      legend.key = element_rect(fill = "white"))
loadings_plot
```



1. Features f3, f4, f5, f6 are positively correlated with PC1 but have little correlation with PC2.
2. Features f7, f8, f9, f10 are positively correlated with both PC1 and PC2.
3. Features f11, f12, f13 are positively correlated with PC2 but negatively correlated with PC1.
4. Features f2, f14, f16 are negatively correlated with PC1 and PC2.
5. Feature f15 is negatively correlated with PC2.
6. Features f6, f8 have a strong impact on PC1 based on the length of the vector. However, f10, f4, f3, f5 have less impact on PC1 even though they all point in the same direction.
7. Features f8, f10, f12 have a strong impact on PC2. However, f9, f11 are less impactful on PC2. Feature f8 and f16, f12 and f4 have negative correlation with each other.

Getting the PCA values

We use the 'get_pca_ind' function to get 3 matrices namely $coord, $cos2, and $contrib. With the help of $coord, we will get the value of the first two principal components. We then create a dataframe containing first two PCA's and Digits in the third column.

```
data_pca_new <- get_pca_ind(pca)
data_pca <- data_pca_new$coord[,c(1,2)]
data_pca <- as.data.frame(data_pca)
```

```r
# Adding the digits column from the factored dataset.
data_pca<- cbind(data_pca, penDigits_factor$digits)
# Renaming the columns
names(data_pca)[1] <- "PC1"
names(data_pca)[2] <- "PC2"
names(data_pca)[3] <- "Digits"
head(data_pca)
```

```
##          PC1         PC2 Digits
## 1 -1.3400161 -1.78180963      8
## 2 -0.9418104 -2.08340573      8
## 3 -4.5516374 -0.01176024      8
## 4  1.6309516  2.36066213      9
## 5  0.9524490  2.14095480      9
## 6  2.2261574 -0.71280779      1
```

Section 2- Custom Palette

Since the digits range from 0 to 9, thus, we had to create 10 different custom colors. Since these digits are unordered qualitative data, thus, we had to create a qualitative palette where each color is distinct, no one color is prominent than another, and the colors should not suggest any sense of order. The steps followed for creating such palette are:

1. First, in the 'I want Hue' tool, I created a basic palette. I chose the H (Hue) value from 13 to 350, since we required 10 different sets of color. I set the C (Chroma) value from 31.5 to 83 so that not much dark shades are there. I took the L (Lightness) value from 35 to 85 to have some bright colors. I chose the option 'hard (Force vector)' so that the color chosen are from the extremes in the cluster and not the average.
2. For some of the colors, which were similar in shades, I found their complementary colors from 'Adobe Color' tool.
3. Then with the help of hcl_color_picker(), I refined some of the colors by varying their saturation and lightness and exported the palette as a vector of Hexadecimals to be used further in the plot.

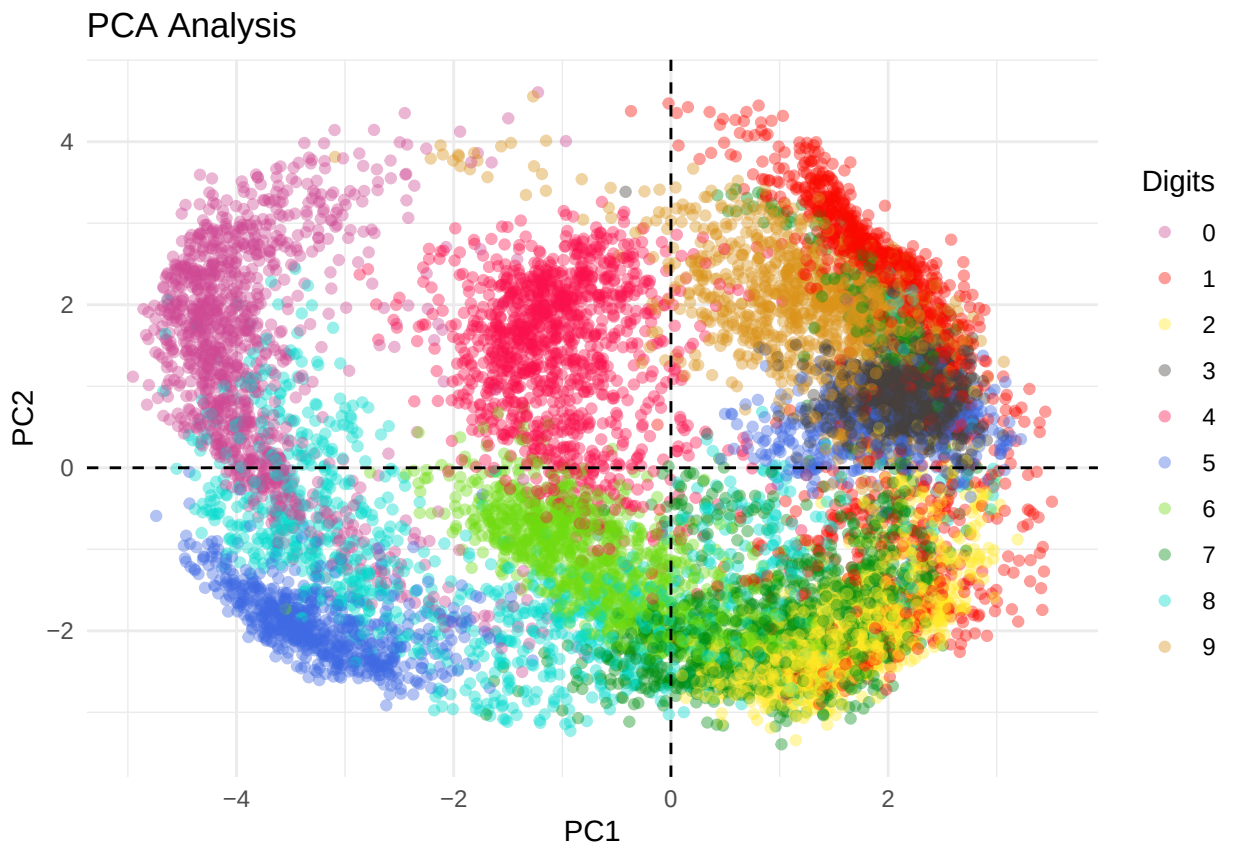
```r
color <- c('#CF4994', '#FA0A00', '#FFE924','#45403D' , '#FA114D','#3F6AE3' ,
                      '#6FDB0F','#008F12', '#00DBCD', '#DB9319')

library(scales)
# To show colors in bar along with hexadecimal.
show_col(color, ncol = 10, borders = NA, cex_label = 0.5)
```

Section 3- Scatterplot

In order to create a scatter plot, we use ggplot2 library. We set our custom colors to represent the digits classified from 0-9. We make use of 'scale_color_manual(values = color)' to set the custom colors.

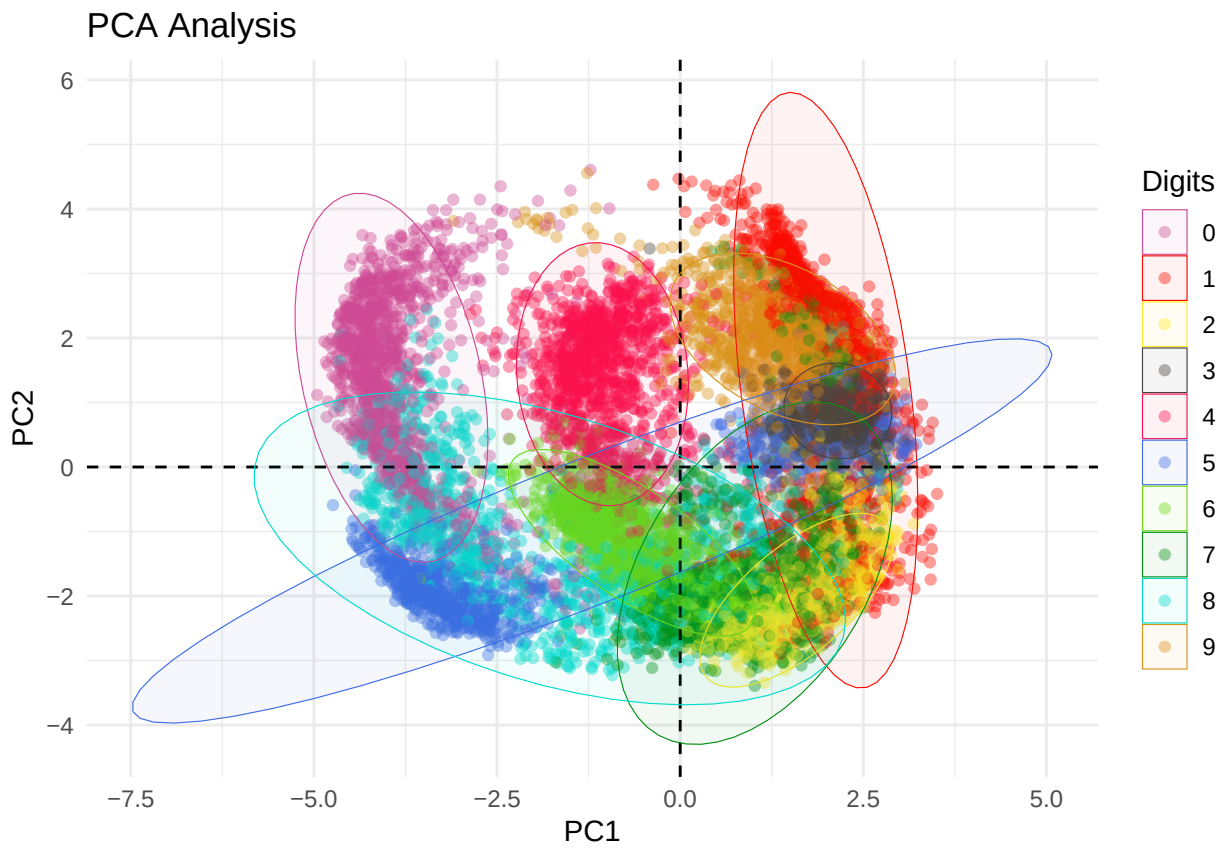
```
library(ggplot2)
ggplot(data_pca, aes(x=PC1, y=PC2, color=Digits)) +
  geom_point(size=1.5, alpha = 0.4) +
  # Using our custom palette for digits
  scale_color_manual(values = color) +
  # Adding the vertical and horizontal line
  geom_vline(xintercept = 0, linetype="dashed") +
  geom_hline(yintercept = 0, linetype="dashed") +
  ggtitle("PCA Analysis") +
  theme_minimal()
```



Using an Ellipse in Scatter plot

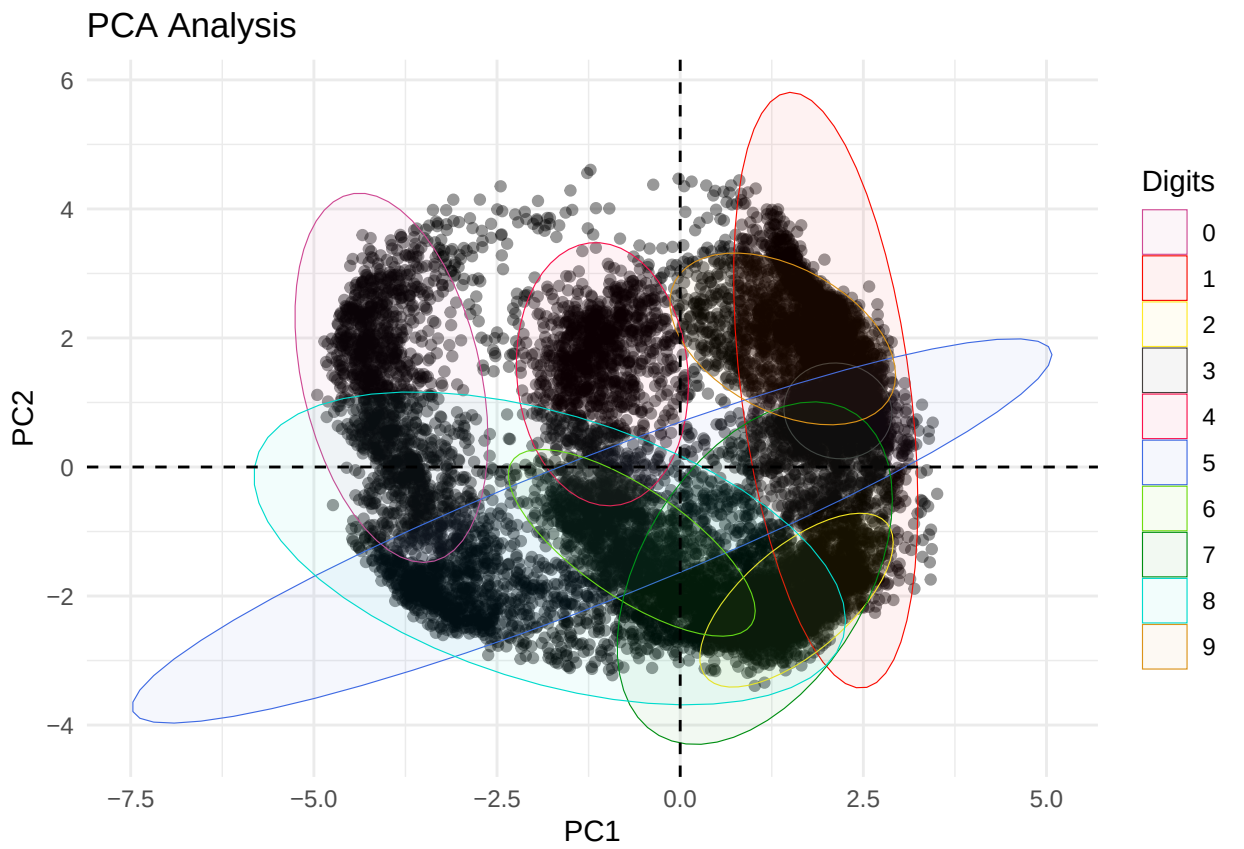
Ellipse can help in better visualize the clusters where there is point overlap. Thus, we make use of 'stat_ellipse' function to draw an ellipse around the points belonging to each Digits value. In order to fill the color of the ellipse, we make use of 'scale_fill_manual' function which fills the custom colors for each ellipse which matches with the color of the group of points.

```
# ellipse with custom colors
pca_scatter<- ggplot(data_pca, aes(x=PC1, y=PC2, color=Digits)) +
  geom_point(size=1.5, alpha = 0.4) +
  # using the custom palette to color the points and the ellipse.
  scale_color_manual(values = color) +
  scale_fill_manual(values = color) +
  # Adding the vertical and horizontal line
  geom_vline(xintercept = 0, linetype="dashed") +
  geom_hline(yintercept = 0, linetype="dashed") +
  # plotting an ellipse for each Digits classified.
  stat_ellipse(geom = "polygon",type = "t",size = 0.2,
               aes(fill = Digits),
               alpha = 0.05) +
  ggtitle("PCA Analysis") +
  theme_minimal()
plot(pca_scatter)
```



After removing the color of data points, the plot looks like as shown below:

```
ggplot(data_pca,aes(x=PC1, y=PC2, colour=Digits)) +
  geom_point(size=1.5, alpha = 0.4, colour="black") +
  stat_ellipse(geom = "polygon",type = "t",size = 0.2,
               aes(fill = Digits),
               alpha = 0.05) +
  scale_color_manual(values = color) +
  scale_fill_manual(values = color) +
  # Adding the vertical and horizontal line
  geom_vline(xintercept = 0, linetype="dashed") +
  geom_hline(yintercept = 0, linetype="dashed") +
  ggtitle("PCA Analysis") +
  theme_minimal()
```



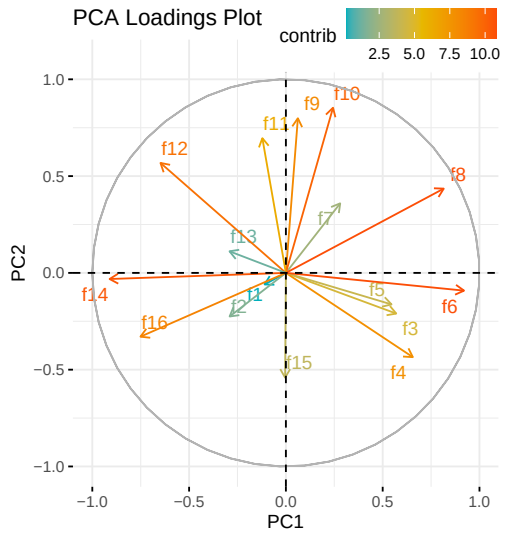
Retaining the colors of data points and ellipse is helpful to the data visualisation. If we make the data points black then there are some points which are not part of the ellipse. In tha case, we won't be able to identify which digits are they classified to.

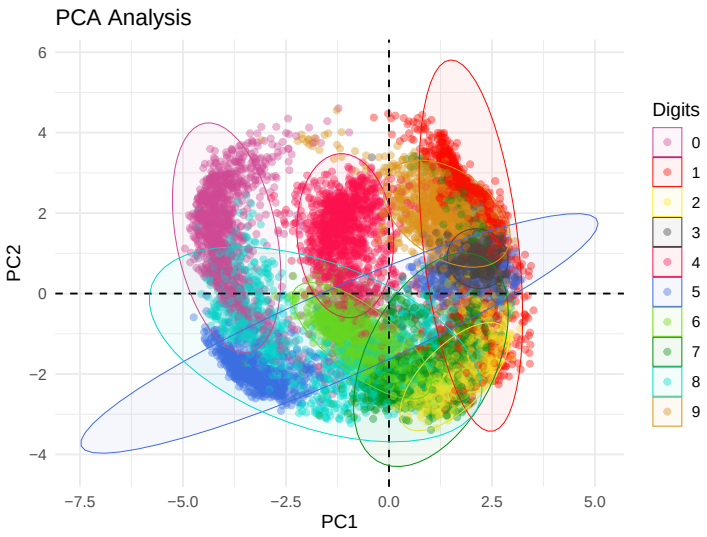
Keepin the color of ellipse also seems significant, as we are able to identify the data points that are distributed across and overlap with many clusters. As we see that in the bottom right corner, there are a lot of overlap of colors of data points, with the help of ellipse, we are able to identify the distribution of the data points of a particular cluster.

Data Points Overlap

1. Since using the scree plot, we found that the first two principal components represent 49.7% of variance of the dataset which is fairly good. If this percentage would have been more, the data points would have been clearly visible with less overlap.
2. The overlap could be more because a simple classifier like K-nearest neighbors was used. Thus, the model classified the digits incorrectly.
3. If we compare loading plot with scatter plot, we can say that if the features are strongly correlated then the overlap will be less.

```
library(patchwork)
loadings_plot + pca_scatter
```

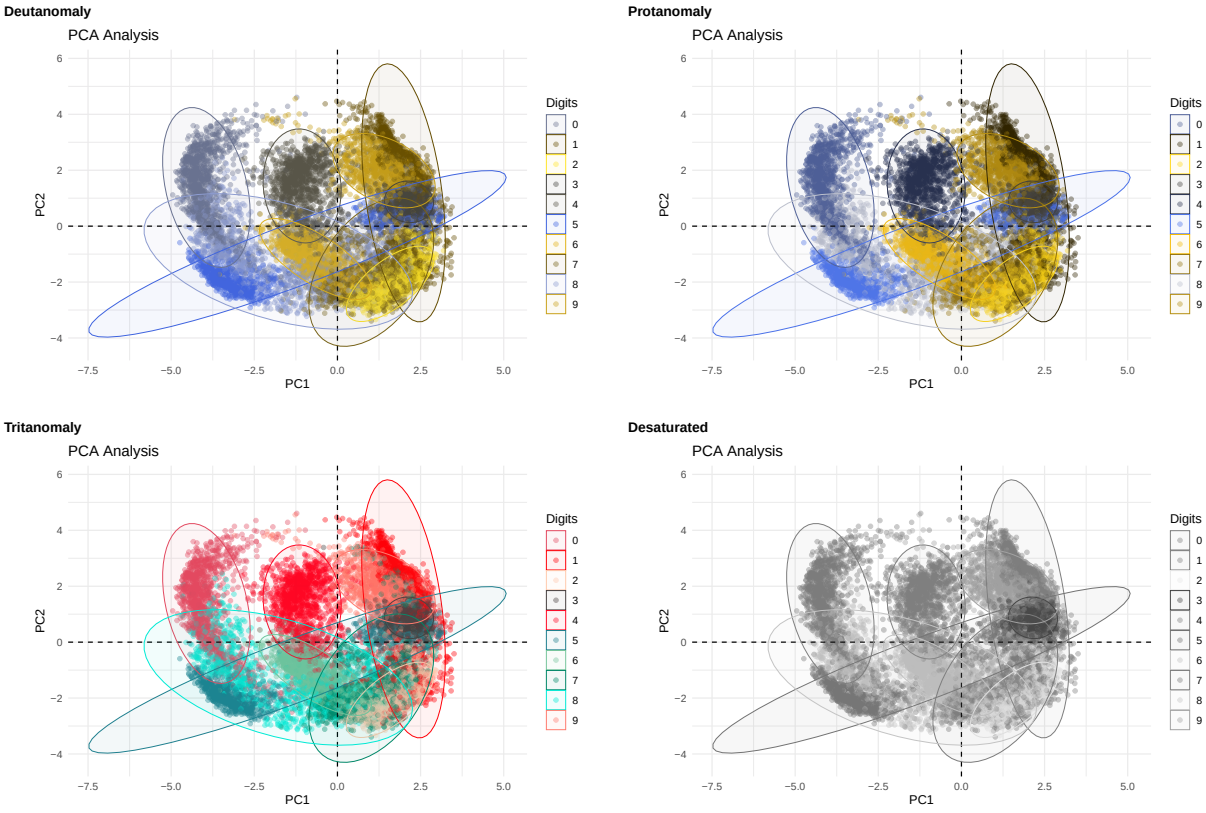




CVD Simulation

The below plot shows the CVD simulation.

```
library(colorblindr)
cvd_grid(pca_scatter)
```



Someone with CVD will be able to identify atleast 8 colors from the scatter plot. It is because I have avoided using Red-Green combination. Such combinations are difficult to read for people with vision deficiency. Instead, I have used different shades of Red like purple, magenta, and orangish red. Also, different colors have different saturation and lightness value. Thus, if a dark red color is used then green used would be light that would help CVD affected people differentiate these colors.