FEBURARY
2024

FATER CHALLENGE

SLDA FINAL PROHJECT 2024

PRESENTED BY
HASSAN MUHAMMAD SIRAJ ZAFAR
MUHAMMAD SAMEER KAZI
FATEMEH GERAEI
SHUJA UR REHMAN

Contents

Introduction

The challenge of estimating store potential for diaper sales in the Naples province is deeply rooted in the retail industry's need for targeted marketing strategies. In this context, the focus is on optimizing product placement and promotional efforts for diapers. This task is especially relevant in scenarios where resource allocation must be maximized due to limited availability, necessitating a strategic approach to enhance store performance and capitalize on untapped market potential.

Case study

This case study involves a comprehensive analysis of various factors that can influence diaper sales across retail outlets in Naples. These factors include:

Socio-Demographic Data:

Understanding the composition of the local population, including gender, and age distribution which can significantly influence diaper demand.

Territorial Features:

Analyzing geographic and environmental characteristics that affect consumer access and shopping behavior, such as store location, area density, and proximity to residential areas.

Points of Interest:

Identifying nearby facilities and services that may impact consumer traffic and shopping patterns, such as parking space, store size, and the type of store- supermarket or hypermarket or so on.

Questions to solve

Addressing this challenge requires answering several complex questions, including:

- How can socio-demographic data be leveraged to predict diaper sales in different areas of Naples? Understanding which demographic factors most strongly correlate with diaper demand can help target marketing efforts more effectively.
- What territorial features are most indicative of high store potential for diaper sales? Determining how geographic location and the physical characteristics of an area influence sales potential can guide the selection of stores for focused marketing campaigns.
- How do points of interest impact the store's potential for diaper sales? Identifying the effect of nearby facilities and services on consumer behavior can help in optimizing store stock and marketing strategies.
- What predictive models and analytical tools can accurately estimate store potential for diaper sales? Developing a reliable method for forecasting sales potential will enable the business to make informed decisions regarding marketing and inventory management.

By addressing these questions, the project aims to develop a strategic framework for enhancing diaper sales across the Naples province, ensuring that marketing efforts are both efficient and effective. This involves not only a deep dive into data analysis but also an innovative approach to integrating socio-demographic insights, territorial mapping, and the identification of strategic points of interest into a cohesive marketing strategy.

Dataset

To achieve this goal, the challenge encompasses analyzing a diverse set of data represented across multiple files, including Dictionary.xlsx, gravitation_NA.csv, shapes_NA.csv, socio_demo_NA.csv, and stores_NA.csv.

1.Dictionary.xlsx:

This file is crucial for understanding the datasets' structure and content. It contains a comprehensive glossary of terms, variables, and their descriptions, including data types and units of measurement. This metadata is essential for accurately interpreting the data across the other files, ensuring consistency and clarity in data analysis.

2.Gravitation_NA.csv:

The gravitation data is expected to quantify the attractiveness or "pull" of each store based on factors that influence consumer visits, such as proximity to key points of interest (POIs), store visibility, and accessibility. This dataset is instrumental in identifying stores with high potential based on geographic and consumer behavior factors.

3.Shapes_NA.csv:

This spatial dataset includes detailed geographic information, such as the coordinates that forms the polygon and the microcode that is assigned to this polygon. Such data is vital for analyzing with the help of Latitude and Longitude where the store lies (In which polygon.

4.Socio_Demo_NA.csv:

Containing socio-demographic information, this file provides insights into the population characteristics around each store, including age distribution. Understanding these demographics is key to estimating diaper sales potential, as it directly relates to the target market's size and purchasing power.

5.Stores_NA.csv: This dataset comprises detailed information about each store, including but not limited to, location coordinates, size, and other operational metrics. Analyzing this data enables the identification of stores with the greatest potential for sales improvement or expansion, guiding targeted marketing and inventory strategies.

By merging the information contained within these files, one can form a comprehensive view of the multifaceted approach needed to estimate store potential for diaper sales. The integration of socio-demographic insights, spatial characteristics, store attributes, and consumer behavior data provides a robust foundation for developing targeted marketing plans that optimize resource allocation and enhance store performance across the Naples province. This multifaceted analysis underscores the importance of leveraging diverse data sources to inform strategic decision-making in retail marketing and sales optimization efforts.

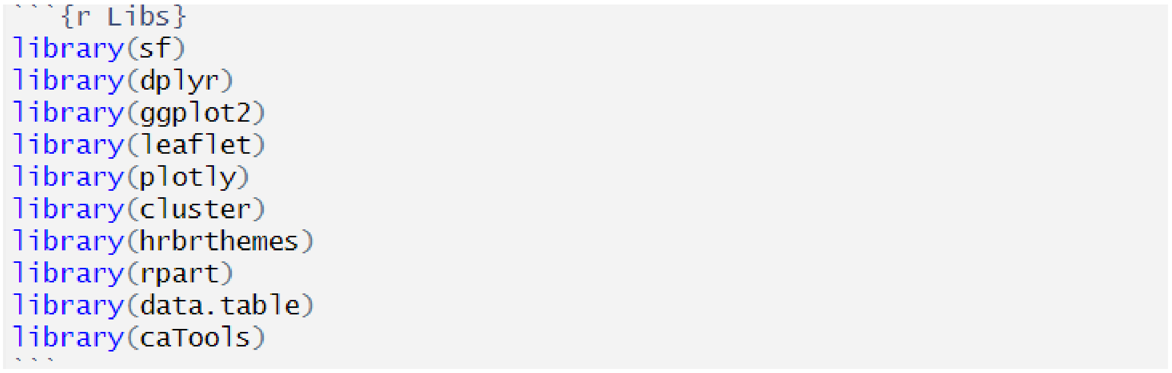
Code Explanation:

Kniting settings

knitr::opts_chunk$set(echo = TRUE)

knitr::opts_chunk$set(warning = FALSE, message = FALSE)

Importing libraries



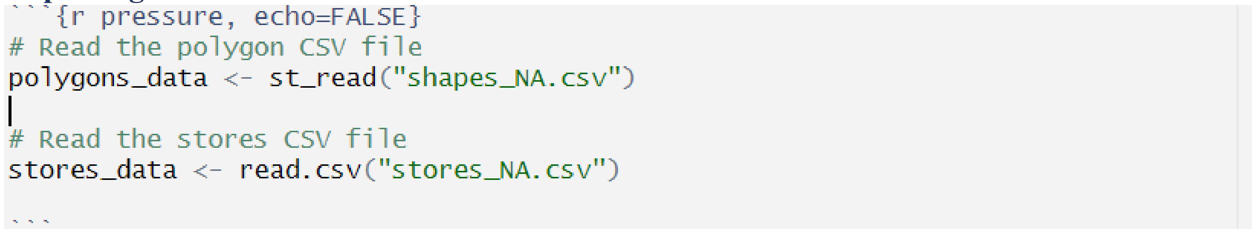
~~~
```{r Libs}
library(sf)
library(dplyr)
library(ggplot2)
library(leaflet)
library(plotly)
library(cluster)
library(hrbrthemes)
library(rpart)
library(data.table)
library(caTools)
```
~~~

- **library(sf)**: This line loads the 'sf' package, which provides functions and classes for working with spatial data structures. It is commonly used for geographic information system (GIS) tasks.
- **library(dplyr)**: This line loads the 'dplyr' package, which provides a set of functions for data manipulation and transformation. It is widely used for tasks such as filtering, selecting, mutating, and summarizing data.
- **library(ggplot2)**: This line loads the 'ggplot2' package, which is a powerful and flexible plotting system in R. It allows users to create complex and customized plots using a grammar of graphics approach.
- **library(leaflet):** This line loads the 'leaflet' package, which provides interactive maps in R. It allows users to create dynamic and interactive maps that can be displayed in web browsers.
- **library(plotly):** This line loads the 'plotly' package, which provides interactive plots in R. It allows users to create and customize interactive visualizations that can be easily shared and embedded in web applications.
- **library(cluster):** This line loads the 'cluster' package, which provides functions for clustering analysis. It includes methods for performing k-means clustering, hierarchical clustering, and other clustering techniques.
- **library(hrbrthemes)**: This line loads the 'hrbrthemes' package, which provides additional themes and styling options for ggplot2 plots. It offers a variety of pre-designed themes for creating visually appealing plots.

- **library(rpart)**: This line loads the 'rpart' package, which provides functions for fitting and visualizing classification and regression trees (CART). It is commonly used for decision tree-based modeling tasks.
- **library(data.table):** This line loads the 'data.table' package, which provides an extension of data frames in R with additional features for fast and efficient data manipulation. It is particularly useful for working with large datasets.
- **library(caTools)**: This line loads the 'caTools' package, which provides functions for data splitting and other utility functions commonly used in machine learning tasks. It includes functions such as sample.split() for creating training and testing datasets.

Data Pre-Processing

Importing data

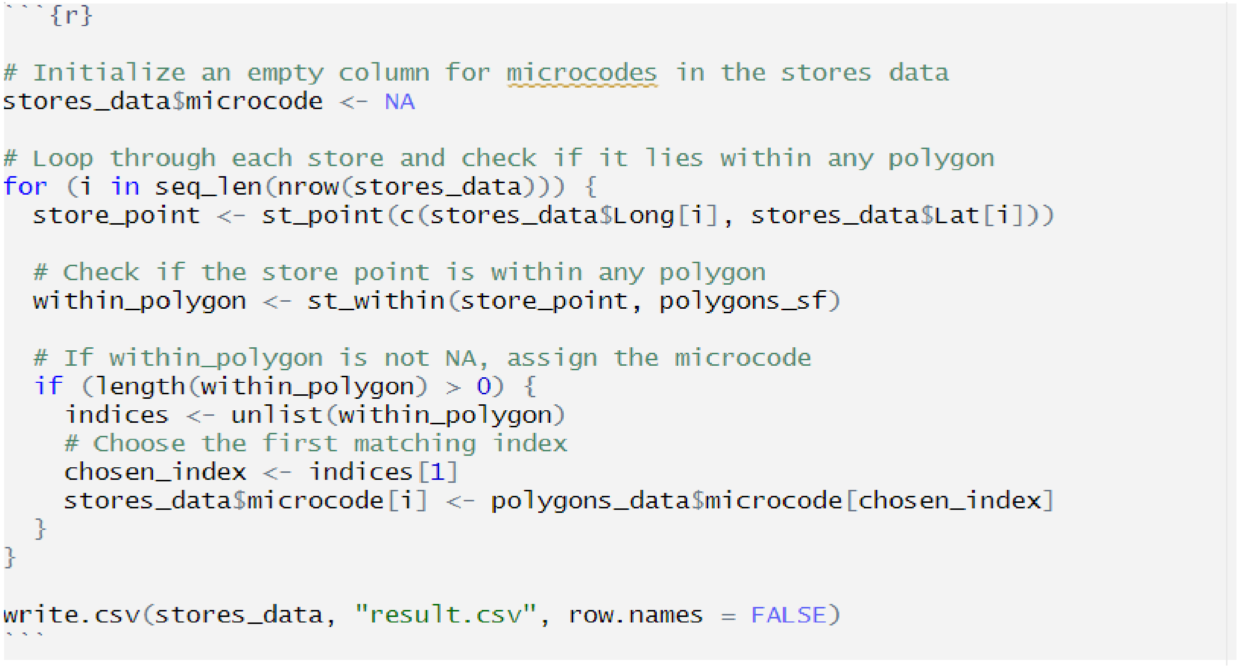


~~~
```{r pressure, echo=FALSE}
# Read the polygon CSV file
polygons_data <- st_read("shapes_NA.csv")

# Read the stores CSV file
stores_data <- read.csv("stores_NA.csv")

```
~~~

The st_read() function from the sf package is used to read the CSV file containing polygon data (shapes_NA.csv) and store it in a data frame (polygons_data).This file likely contains information about geographic regions represented as polygons.

The read.csv() function is used to read the CSV file containing store data (stores_NA.csv) and store it in a data frame (stores_data).This file likely contains information about store locations, including latitude, longitude, store ID, store name, and potential.

Assigning microcode to store



~~~
```{r}

# Initialize an empty column for microcodes in the stores data
stores_data$microcode <- NA

# Loop through each store and check if it lies within any polygon
for (i in seq_len(nrow(stores_data))) {
  store_point <- st_point(c(stores_data$Long[i], stores_data$Lat[i]))

  # Check if the store point is within any polygon
  within_polygon <- st_within(store_point, polygons_sf)

  # If within_polygon is not NA, assign the microcode
  if (length(within_polygon) > 0) {
    indices <- unlist(within_polygon)
    # Choose the first matching index
    chosen_index <- indices[1]
    stores_data$microcode[i] <- polygons_data$microcode[chosen_index]
  }
}

write.csv(stores_data, "result.csv", row.names = FALSE)
```
~~~

This code snippet performs the following tasks:

Initialize an empty column for microcodes:

It adds a new column named "microcode" to the stores_data data frame and initializes it with NA values.

Loop through each store and check if it lies within any polygon:

It iterates through each row of the stores_data data frame using a for loop.For each store, it creates a point geometry representing the store's location using the st_point() function from the sf package.

It then checks if the store point lies within any polygon in the polygons_sf spatial object (converted from polygons_data using st_as_sf() earlier) using the st_within() function.

If the store point is within a polygon, it retrieves the corresponding microcode from the polygons_data data frame and assigns it to the store's "microcode" column in stores_data.

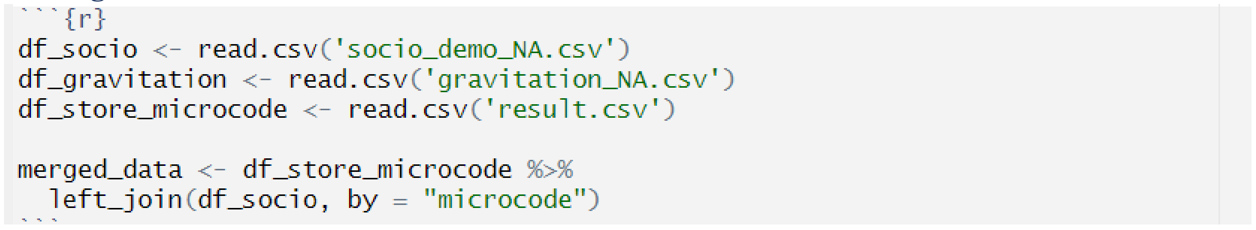
The microcode is assigned based on the first matching polygon index.

Write the result to a CSV file:

It writes the updated stores_data data frame to a CSV file named "result.csv" using the write.csv() function. The row.names = FALSE argument ensures that row names are not included in the output file.

This code essentially performs a spatial join between the store locations and the polygons based on their spatial relationship (i.e., whether a store is located within a polygon). It assigns a "microcode" to each store based on the polygon it lies within.The resulting CSV file "result.csv" will contain the original store data along with the newly added "microcode" column, indicating the corresponding microcode for each store based on its location within the polygons.

Joining data



```{r}
df_socio <- read.csv('socio_demo_NA.csv')
df_gravitation <- read.csv('gravitation_NA.csv')
df_store_microcode <- read.csv('result.csv')

merged_data <- df_store_microcode %>%
  left_join(df_socio, by = "microcode")
```

Read CSV files:

df_socio <- read.csv('socio_demo_NA.csv'): Reads the data from the CSV file named 'socio_demo_NA.csv' and stores it in the data frame df_socio.

df_gravitation <- read.csv('gravitation_NA.csv'): Reads the data from the CSV file named 'gravitation_NA.csv' and stores it in the data frame df_gravitation.

df_store_microcode <- read.csv('result.csv'): Reads the data from the CSV file named 'result.csv' and stores it in the data frame df_store_microcode.

Merge data frames:

merged_data <- df_store_microcode %>% left_join(df_socio, by = "microcode"): This line uses the %>% (pipe) operator from the dplyr package to perform a left join operation between df_store_microcode and df_socio data frames based on the common column "microcode".

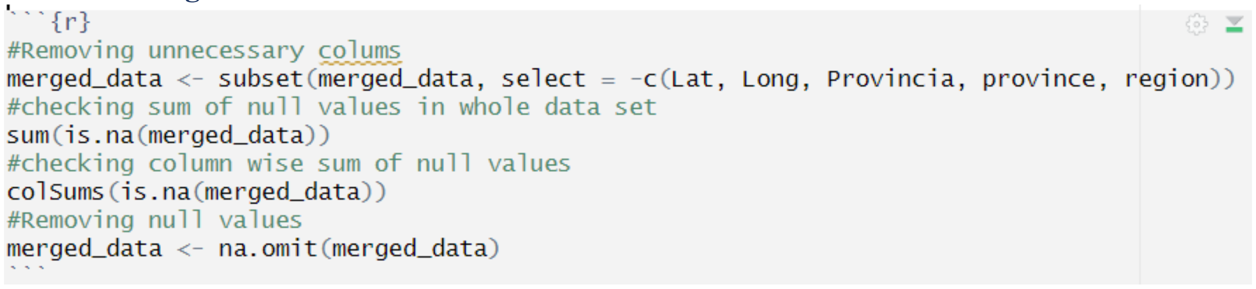
A left join keeps all the rows from the left data frame (df_store_microcode) and matches rows from the right data frame (df_socio) based on the values in the "microcode" column.The result of the join operation is stored in the data frame merged_data.

This code is likely part of a data integration or data merging process, where different datasets (in this case, socio-demographic data and store microcode data) are combined based on a common key (in this case, "microcode").

By performing a left join, it ensures that all rows from the store microcode data are retained, and socio-demographic data is added to it where available (based on matching microcode values).

The resulting merged_data data frame will contain the combined information from both datasets, allowing for further analysis or modeling that may require socio-demographic information alongside store microcode data.

Data Cleaning



~~~
```{r}
#Removing unnecessary colums
merged_data <- subset(merged_data, select = -c(Lat, Long, Provincia, province, region))
#checking sum of null values in whole data set
sum(is.na(merged_data))
#checking column wise sum of null values
colSums(is.na(merged_data))
#Removing null values
merged_data <- na.omit(merged_data)
```
~~~

Removing Unnecessary Columns:

This line removes specific columns (Lat, Long, Provincia, province, region) from the merged_data dataframe using the subset() function. The select argument specifies the columns to keep, while the -c() notation indicates the columns to exclude.

Checking Sum of Null Values in the Whole Dataset:

This code calculates the total number of missing values (NA) across the entire merged_data dataframe using the is.na() function to identify NA values and sum() function to count them.

Checking Column-wise Sum of Null Values:

This line calculates the sum of missing values for each column in the merged_data dataframe using is.na() to identify NA values and colSums() to sum them column-wise.

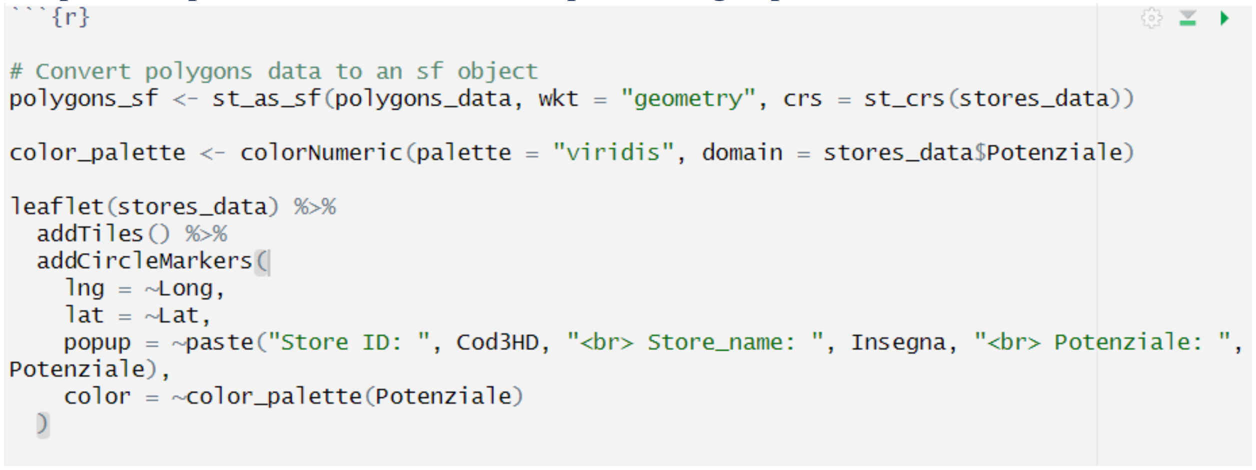
Removing Null Values:

This code removes rows containing any NA values from the **merged_data** dataframe using the **na.omit()** function. Rows with NA values in any column are removed entirely from the dataset.

We removed unnecessary variables to simplify the dataset and focus the analysis on relevant variables.Removing rows with missing values ensures that the analysis is based on complete data and avoids potential biases or inaccuracies introduced by missing values.

Exploratory Data Analysis

Graphicall representaion of Stores on map according to potential



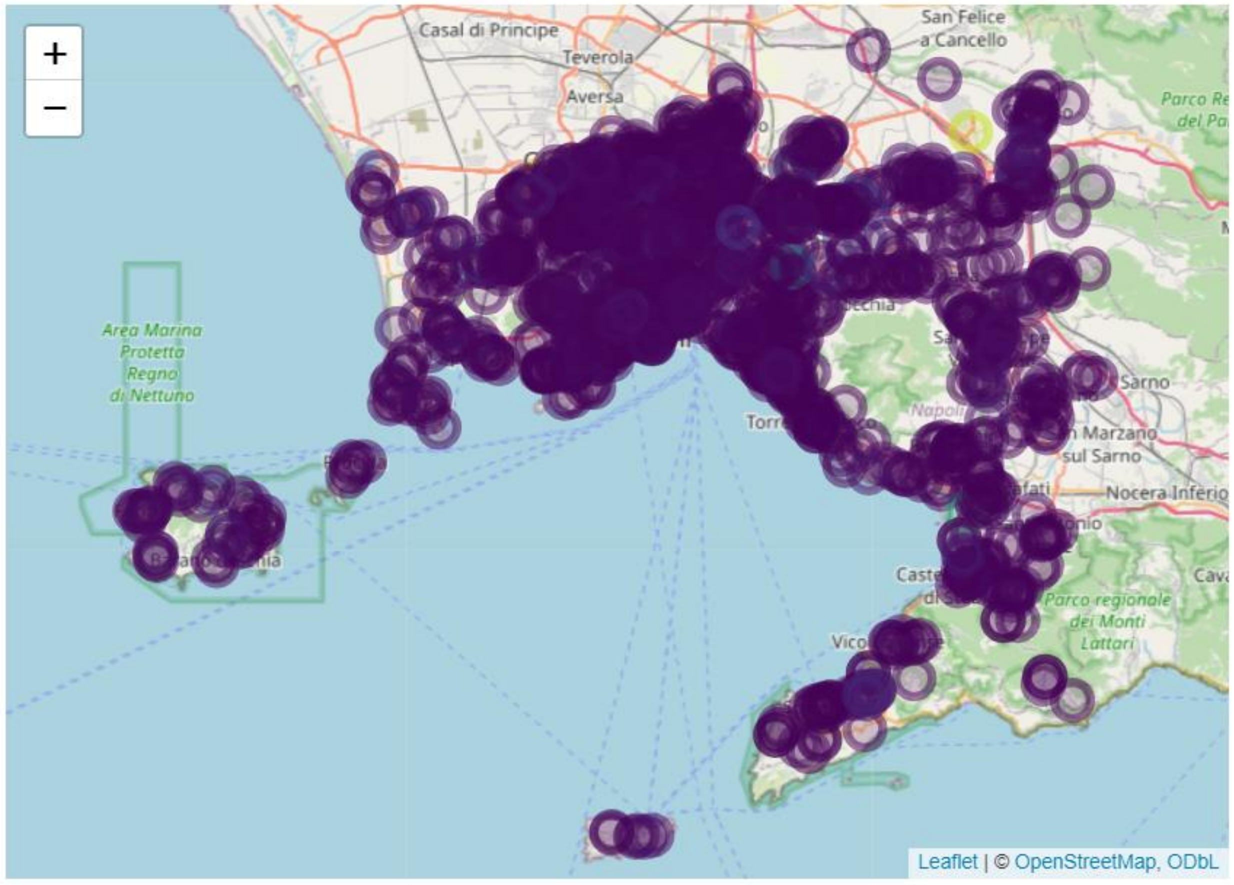
~~~r
```{r}

# Convert polygons data to an sf object
polygons_sf <- st_as_sf(polygons_data, wkt = "geometry", crs = st_crs(stores_data))

color_palette <- colorNumeric(palette = "viridis", domain = stores_data$Potenziale)

leaflet(stores_data) %>%
  addTiles() %>%
  addCircleMarkers(
    lng = ~Long,
    lat = ~Lat,
    popup = ~paste("Store ID: ", Cod3HD, "<br> Store_name: ", Insegna, "<br> Potenziale: ",
Potenziale),
    color = ~color_palette(Potenziale)
  )
~~~

This code segment demonstrates two different approaches for visualizing store locations: using Leaflet for interactive maps.

The Leaflet approach creates an interactive map with circle markers representing store locations, and clicking on markers displays additional information about each store.

These visualizations can be useful for exploring the spatial distribution of stores and their potential, aiding in decision-making and analysis.

Otuput:



Box plots:

~~~
```{r}
ColNames <- c("population", "Potenziale", "MQVEND")

# Loop through each column and create a box plot
for (col in ColNames) {
  plotly_plot <- plot_ly(data = merged_data,
                         type = "box",
                         x = ~get(col),
                         name = col,
                         marker = list(color = "red"),
                         line = list(color = "yellow")) %>%
    layout(title = paste("Box Plot of", col),
           xaxis = list(title = col))

  # Display the interactive plot
  print(plotly_plot)
}
```
~~~

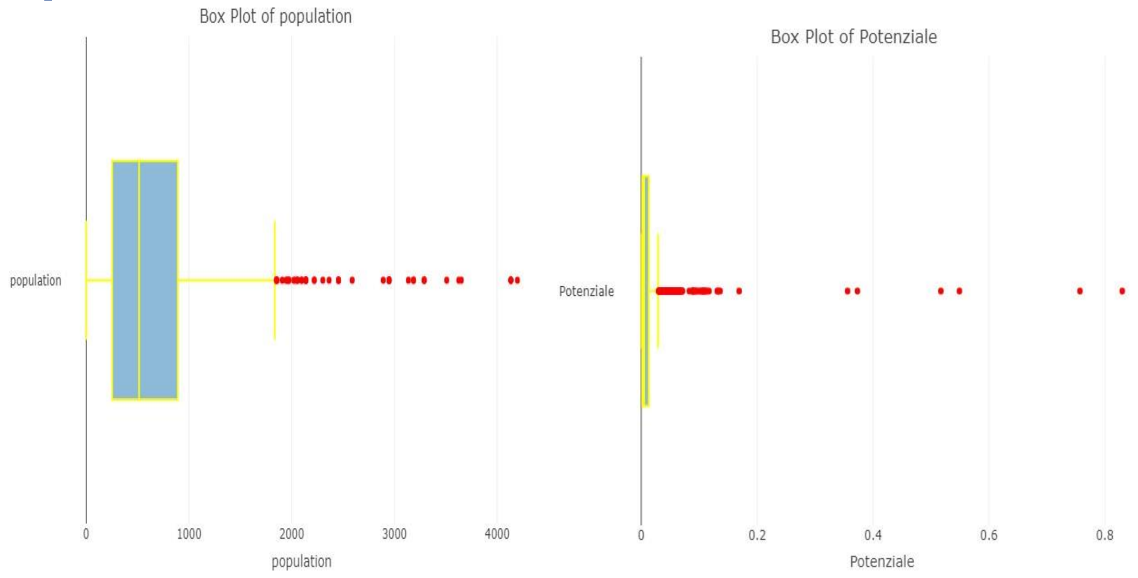
This code dynamically generates box plots for each specified column in merged_data.

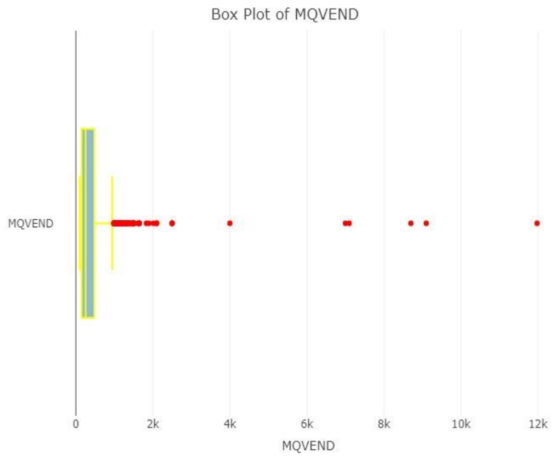
It utilizes the plot_ly() function from the plotly package to create interactive box plots.
Each box plot represents the distribution of values in a particular column, with outliers, quartiles, and median visualized.The loop ensures that a box plot is created for each column specified in ColNames.

The interactive nature of the plots allows users to explore the data further by hovering over data points, zooming, and panning.

This used to display key summary statistics such as the median, quartiles, and identify outliers.

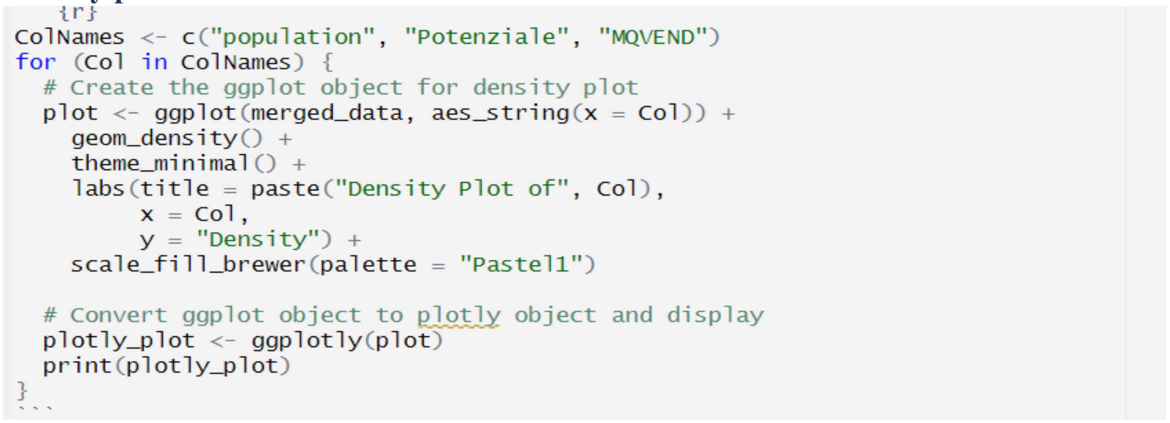
Output:





As evident from the graph above there are multiple outliers in potenziale which may affect our analysis.

Density plot



```
  {r}
ColNames <- c("population", "Potenziale", "MQVEND")
for (Col in ColNames) {
  # Create the ggplot object for density plot
  plot <- ggplot(merged_data, aes_string(x = Col)) +
    geom_density() +
    theme_minimal() +
    labs(title = paste("Density Plot of", Col),
         x = Col,
         y = "Density") +
    scale_fill_brewer(palette = "Pastel1")

  # Convert ggplot object to plotly object and display
  plotly_plot <- ggplotly(plot)
  print(plotly_plot)
}
```
```
```

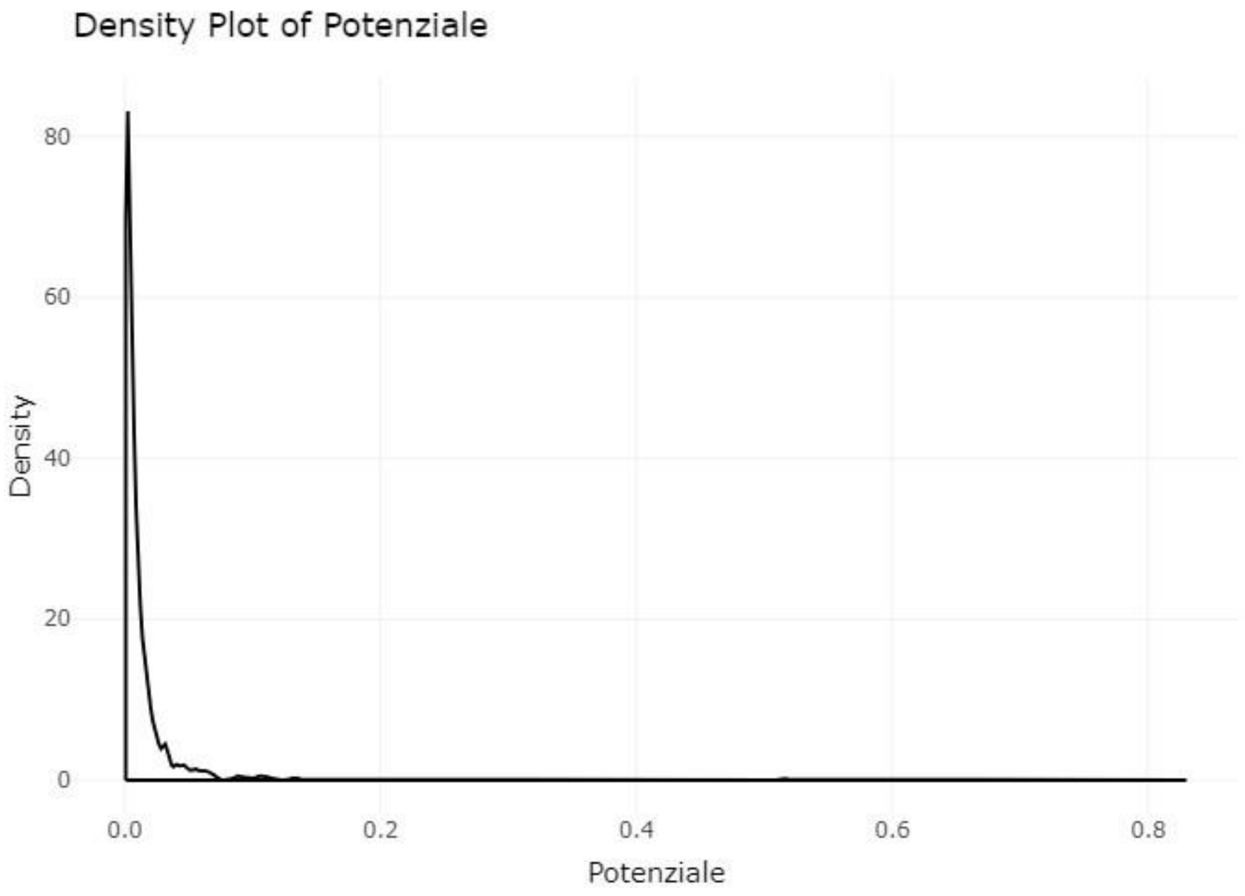
This code dynamically generates density plots for each specified column in the dataset.

Density plots provide insights into the distribution of values within each column, helping to identify patterns, skewness, and multimodality in the data.

The use of ggplotly() converts each static ggplot object into an interactive plotly object, allowing users to interactively explore the density plots (e.g., hover over data points for details, zoom in or out, pan across the plot).

By visualizing the distribution of each variable, analysts can gain a better understanding of the data and make informed decisions during the exploratory data analysis process.

Output:



All the graph suggests that the columns checked are negatively skewed.

Normality testing

~~~
```{r}
ks.test(unique(merged_data$population), "pnorm")

ks.test(unique(merged_data$Potenziale), "pnorm")

ks.test(unique(merged_data$MQVEND), "pnorm")
```
~~~

The Kolmogorov-Smirnov (KS) test is a non-parametric statistical test used to assess whether a sample comes from a specific distribution. It compares the empirical cumulative distribution function (ECDF) of the data with the cumulative distribution function (CDF) of a reference distribution. The test statistic (D) represents the maximum vertical deviation between the two distribution functions.

For each column (population, Potenziale and MQVEND) test is ran to check if the data is normally distributed.

A value of p-value for all three rejects the null hypothesis and hence we can say that the data is not normally distributed.

Output:

```
ks.test(unique(merged_data$population), "pnorm")
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  unique(merged_data$population)
## D = 0.99723, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

```
ks.test(unique(merged_data$Potenziale), "pnorm")
```

```
##
##  Exact one-sample Kolmogorov-Smirnov test
##
## data:  unique(merged_data$Potenziale)
## D = 0.5004, p-value = 9.992e-16
## alternative hypothesis: two-sided
```

```
ks.test(unique(merged_data$MQVEND), "pnorm")
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  unique(merged_data$MQVEND)
## D = 1, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

For each test, the small p-values indicate strong evidence against the null hypothesis of the variable following the expected distribution. The specific nature of the difference would depend on the context of your analysis and the characteristics of your data.

Removing Outliers

```
outlier_threshold <- 1.5
iqr_value <- IQR(merged_data$Potenziale)

print(quantile(merged_data$Potenziale)[2] - outlier_threshold * iqr_value)
print(quantile(merged_data$Potenziale)[4] + outlier_threshold * iqr_value)
# Filter out rows with values outside the range (1.5 times IQR)
merged_data <- merged_data %>%
  filter(between(Potenziale, quantile(Potenziale)[2] - outlier_threshold * iqr_value,
quantile(Potenziale)[4] + outlier_threshold * iqr_value))
```

The code aims to remove outliers from the Potenziale variable in the merged_data dataframe using the IQR method.Outliers are identified as values lying outside the range defined by 1.5 times the IQR above the third quartile (Q3) and below the first quartile (Q1).After executing this code, merged_data will contain only the rows where Potenziale values are within the specified outlier bounds, effectively removing outliers from the dataset.

Output:

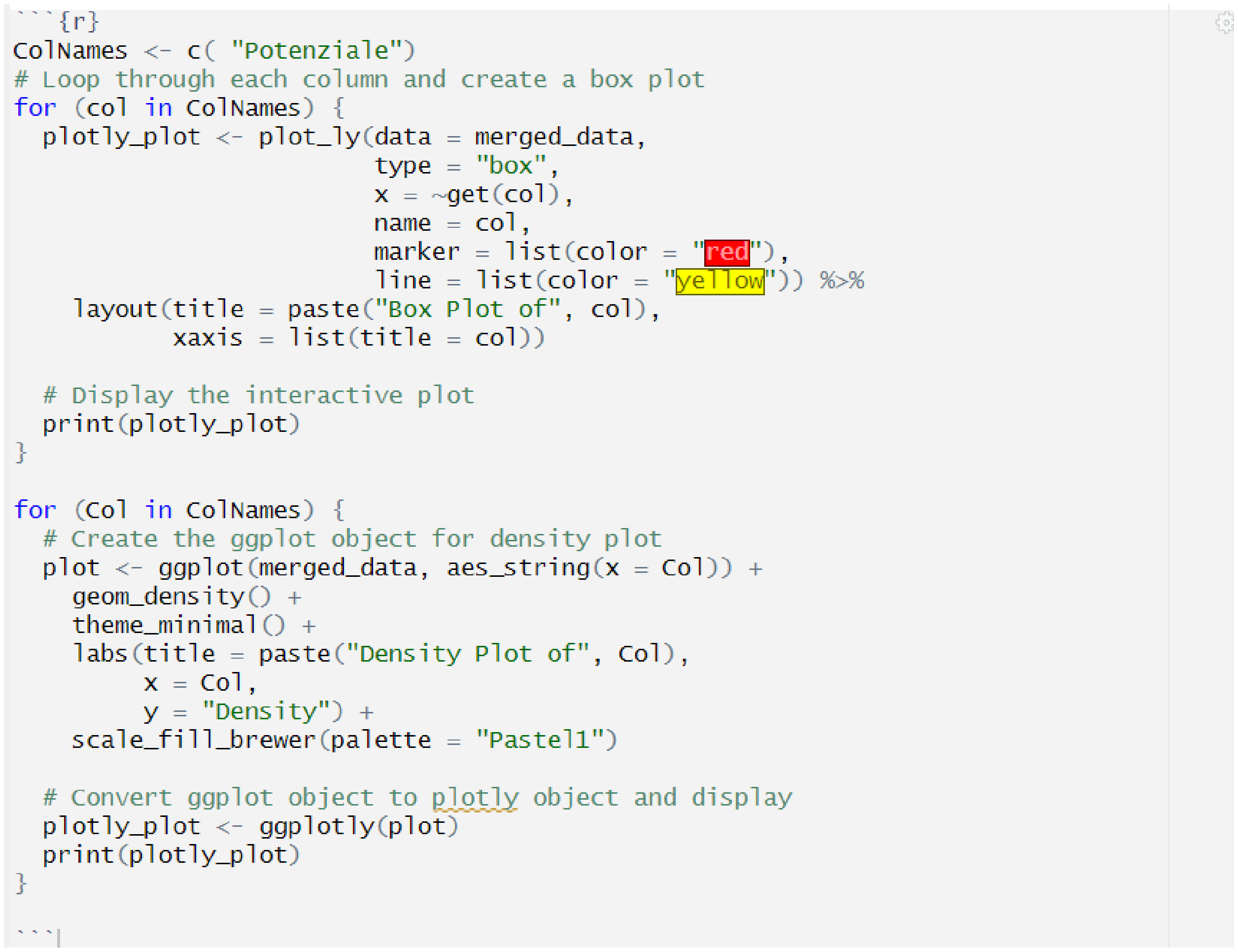
```
print(quantile(merged_data$Potenziale)[2] - outlier_threshold * iqr_value)
```

```
##      25%
## -0.0145
```

```
print(quantile(merged_data$Potenziale)[4] + outlier_threshold * iqr_value)
```

```
##    75%
## 0.0295
```

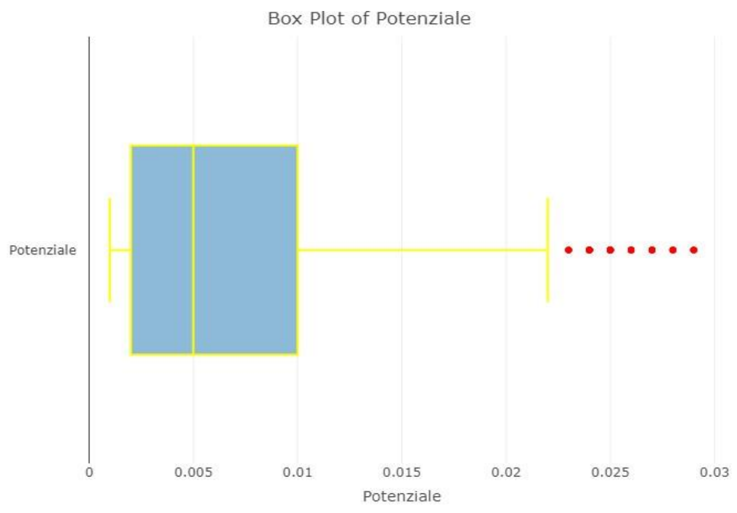
Box and density plot after removing outliers



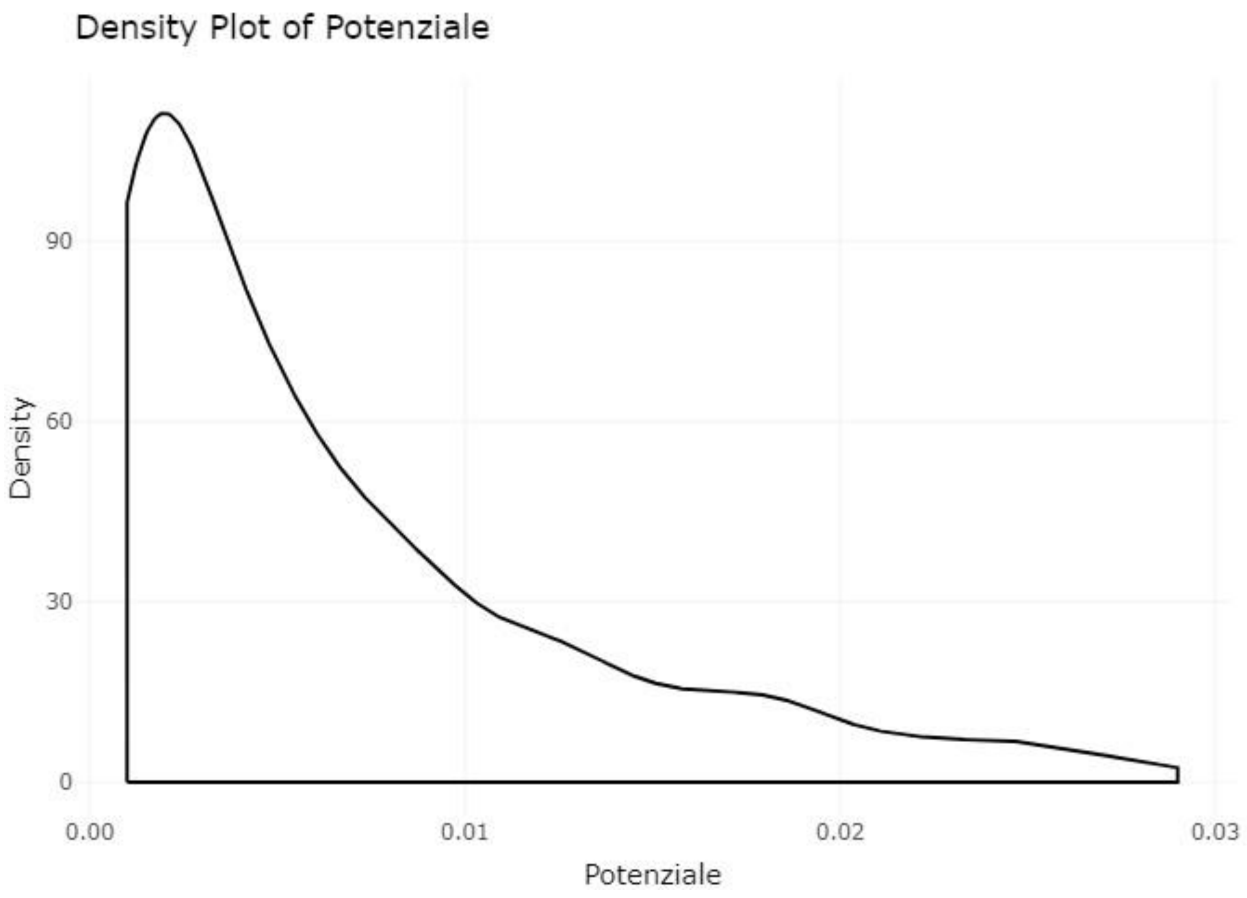
~~~
```{r}
ColNames <- c( "Potenziale")
# Loop through each column and create a box plot
for (col in ColNames) {
  plotly_plot <- plot_ly(data = merged_data,
                         type = "box",
                         x = ~get(col),
                         name = col,
                         marker = list(color = "red"),
                         line = list(color = "yellow")) %>%
    layout(title = paste("Box Plot of", col),
           xaxis = list(title = col))

  # Display the interactive plot
  print(plotly_plot)
}

for (Col in ColNames) {
  # Create the ggplot object for density plot
  plot <- ggplot(merged_data, aes_string(x = Col)) +
    geom_density() +
    theme_minimal() +
    labs(title = paste("Density Plot of", Col),
         x = Col,
         y = "Density") +
    scale_fill_brewer(palette = "Pastel1")

  # Convert ggplot object to plotly object and display
  plotly_plot <- ggplotly(plot)
  print(plotly_plot)
}

```
~~~

the is we again visualized a density plot and a bar plot subsequent to the removal of outliers from the dataset.

Output



The box plot and density plot suggest that outliers are now minimized and we are safe to perform further analysis

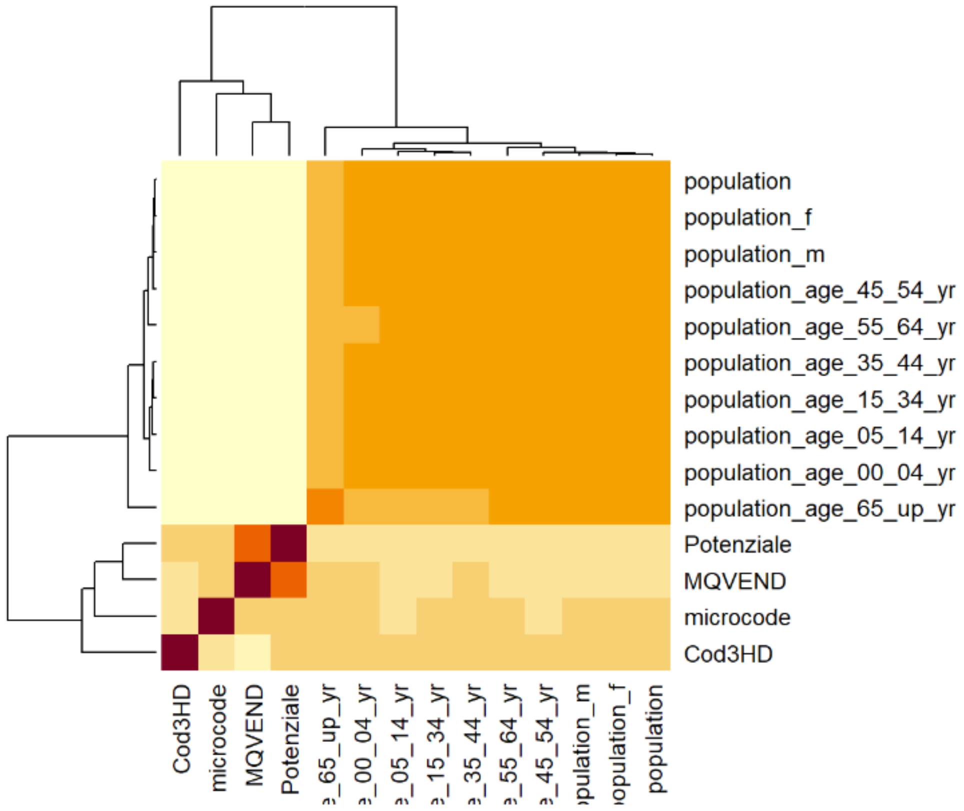


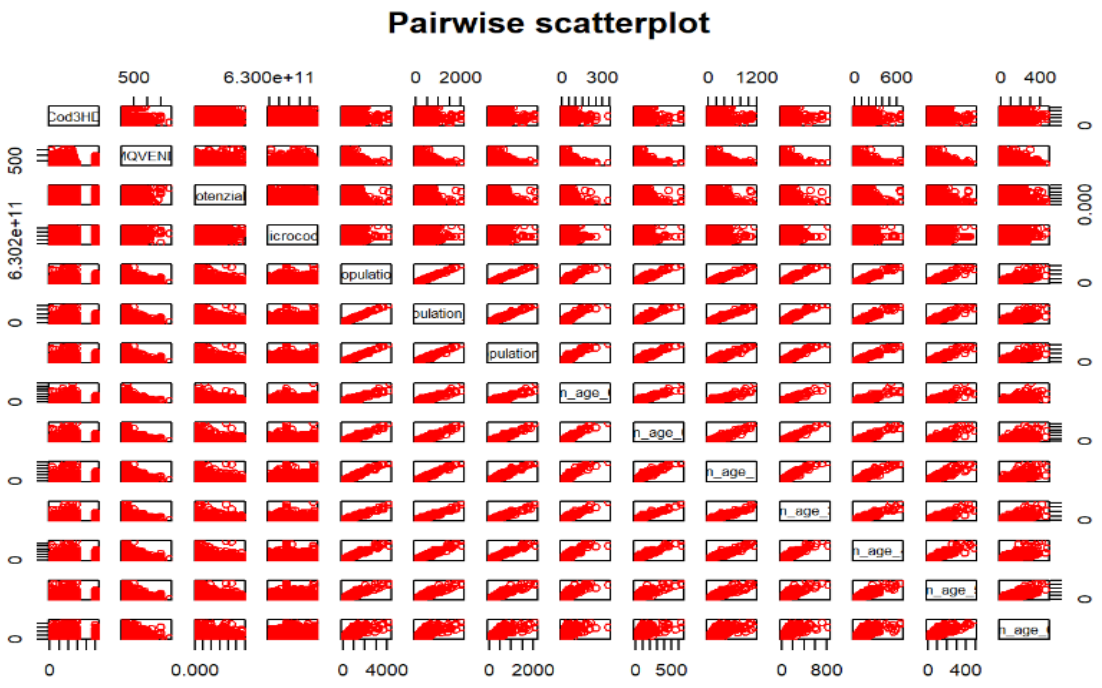
Correlation heatmap and PairPlot

~~~
```{r}
heatmap(cor(select(merged_data, where(is.numeric))))
pairs(select(merged_data, where(is.numeric)),
            main = "Pairwise scatterplot",
            col = "red")


```
~~~

Both the heatmap and pairwise scatterplot provide complementary insights into the relationships between variables in a dataset, with the heatmap focusing on correlation coefficients and the pairwise scatterplot focusing on visual patterns of association.

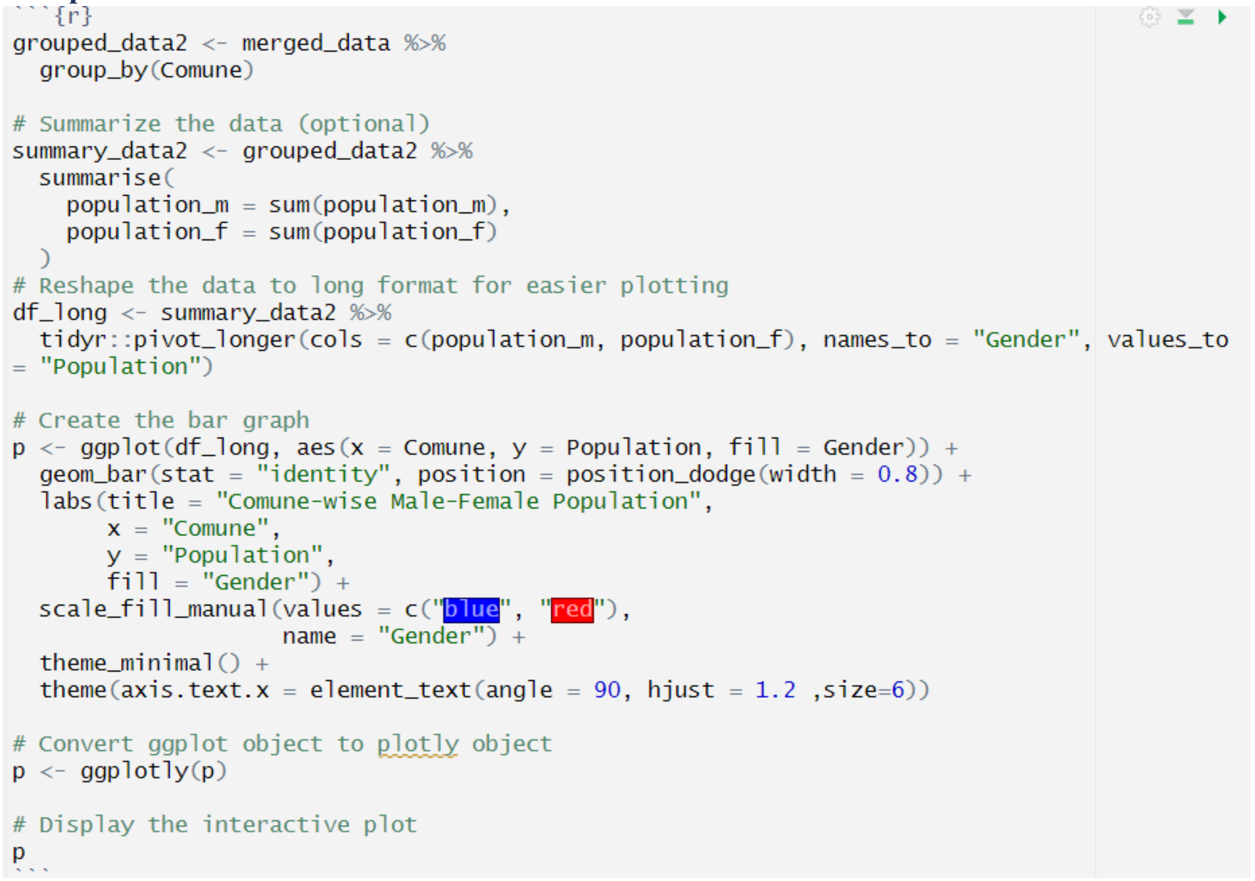
Output:


population
population_f
population_m
population_age_45_54_yr
population_age_55_64_yr
population_age_35_44_yr
population_age_15_34_yr
population_age_05_14_yr
population_age_00_04_yr
population_age_65_up_yr
Potenziale
MQVEND
microcode
Cod3HD


Pairwise scatterplot
Cod3HD
MQVEND
Potenziale
microcode

pair graph there is no significant relationship between Potenziale and any other variable except size of store (MQVEND) column with highest coorelation.

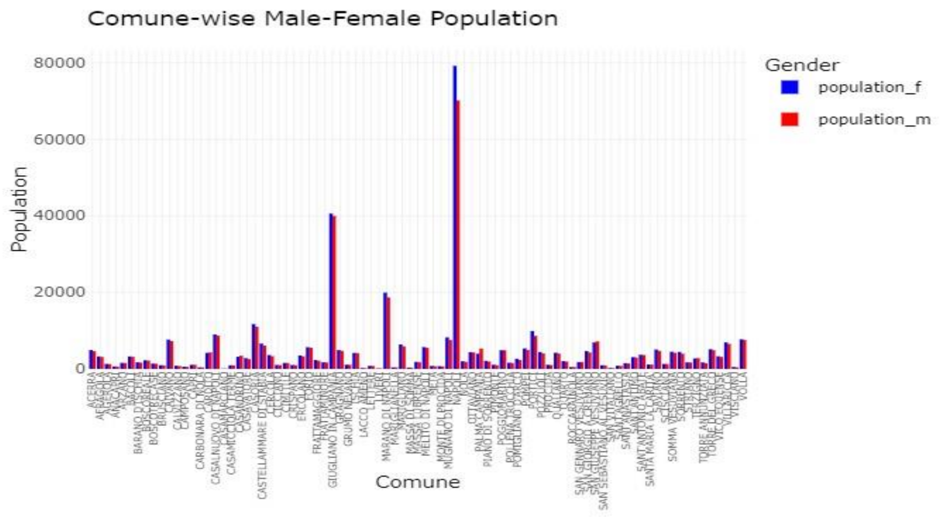
Comparision between male and female data



~~~
```{r}
grouped_data2 <- merged_data %>%
  group_by(Comune)

# Summarize the data (optional)
summary_data2 <- grouped_data2 %>%
  summarise(
    population_m = sum(population_m),
    population_f = sum(population_f)
  )
# Reshape the data to long format for easier plotting
df_long <- summary_data2 %>%
  tidyr::pivot_longer(cols = c(population_m, population_f), names_to = "Gender", values_to
= "Population")

# Create the bar graph
p <- ggplot(df_long, aes(x = Comune, y = Population, fill = Gender)) +
  geom_bar(stat = "identity", position = position_dodge(width = 0.8)) +
  labs(title = "Comune-wise Male-Female Population",
        x = "Comune",
        y = "Population",
        fill = "Gender") +
  scale_fill_manual(values = c("blue", "red"),
                     name = "Gender") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1.2 ,size=6))

# Convert ggplot object to plotly object
p <- ggplotly(p)

# Display the interactive plot
p
```
~~~

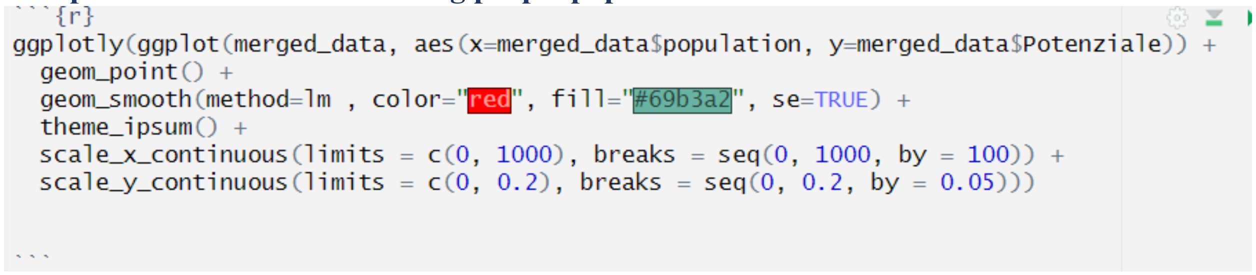
This code calculates the sum of male and female Population for each area (Comune) in the merged_data dataframe.This code provides an interactive visualization of the male and female population distribution across different areas, to explore and compare population demographics dynamically.

Output:



This comparison was done to look for kind of areas where the gender wise difference is maybe impacting the market but from the graph it was evident that most of the areas have similar numbers of man and woman.

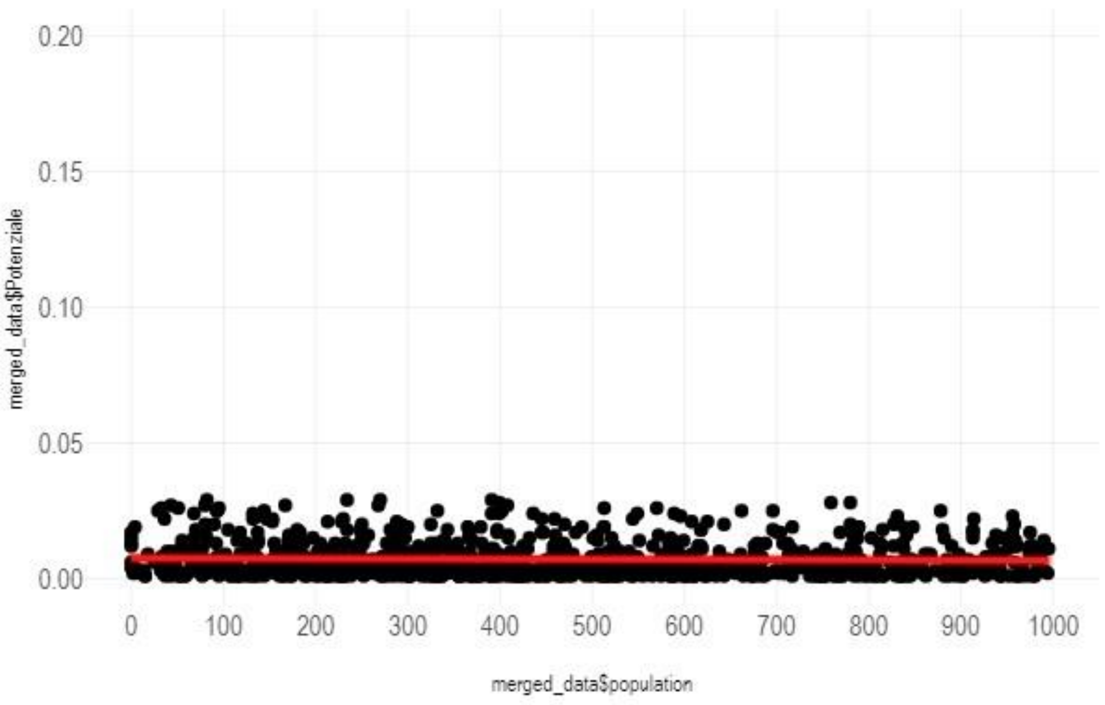
Comparision of Potential and Avg people population



~~~
```{r}
ggplotly(ggplot(merged_data, aes(x=merged_data$population, y=merged_data$Potenziale)) +
  geom_point() +
  geom_smooth(method=lm , color="red", fill="#69b3a2", se=TRUE) +
  theme_ipsum() +
  scale_x_continuous(limits = c(0, 1000), breaks = seq(0, 1000, by = 100)) +
  scale_y_continuous(limits = c(0, 0.2), breaks = seq(0, 0.2, by = 0.05)))


```
~~~

this code creates an interactive scatter plot with a linear regression line fitted to the data, providing insights into the relationship between the population and Potenziale variables in the merged_data dataframe.

Output:



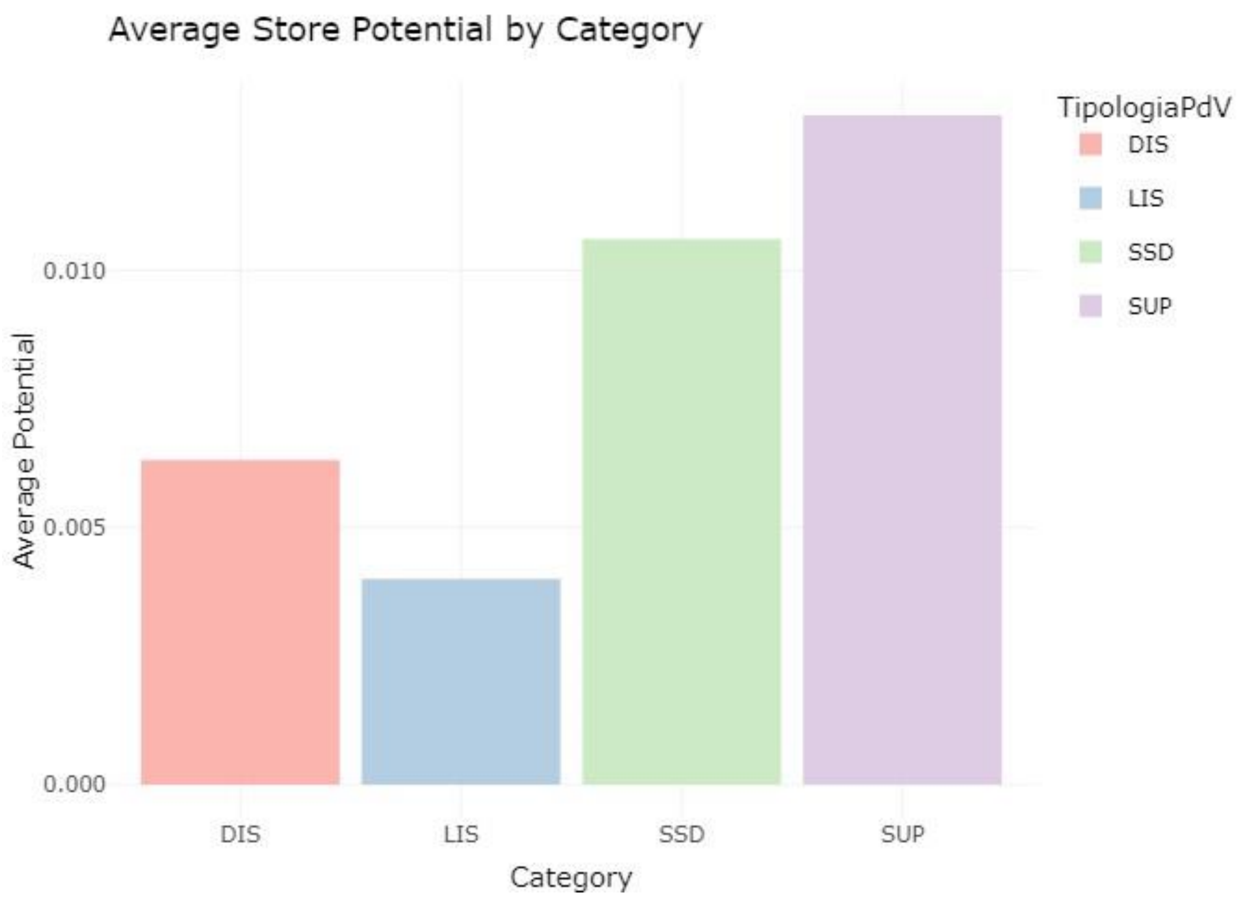
This was an attempt to look closer into the relationship between potential of a store with respect to population in that area but apparently standalone population does not impact the potential of the store whatsoever.

Average Store Potential by store Category

~~~
```{r}

average_potential <- merged_data %>%
  group_by(TipologiaPdV) %>%
  summarise(AveragePotential = mean(Potenziale))

ggplotly(ggplot(average_potential, aes(x = TipologiaPdV, y = AveragePotential, fill =
TipologiaPdV)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Average Store Potential by Category",
       x = "Category",
       y = "Average Potential") +
  scale_fill_brewer(palette = "Pastel1"))


```
~~~

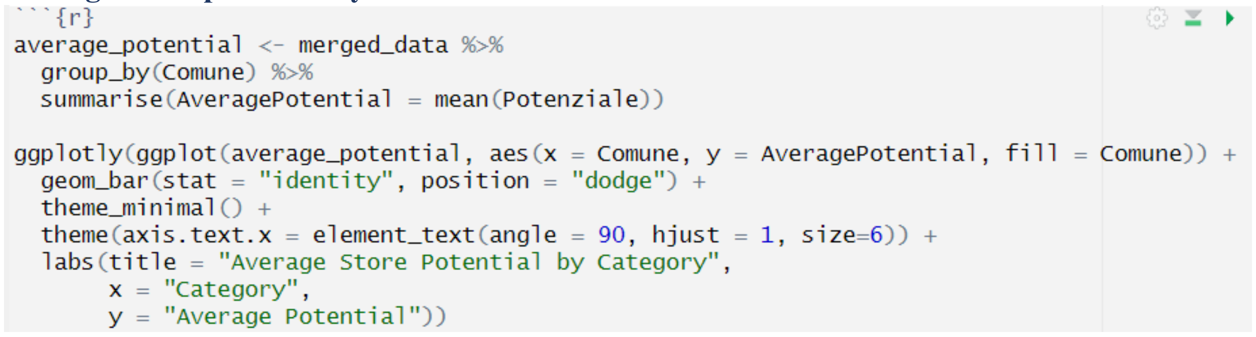
This code calculates the average potential (Potenziale) for each store category (TipologiaPdV) in the merged_data dataframe and then creates an interactive bar plot using ggplotly()

Output:



This graph tells us that most of the potential stores are supermarket along with drugstore taking second place following by discount centers and at last Libero Servizio

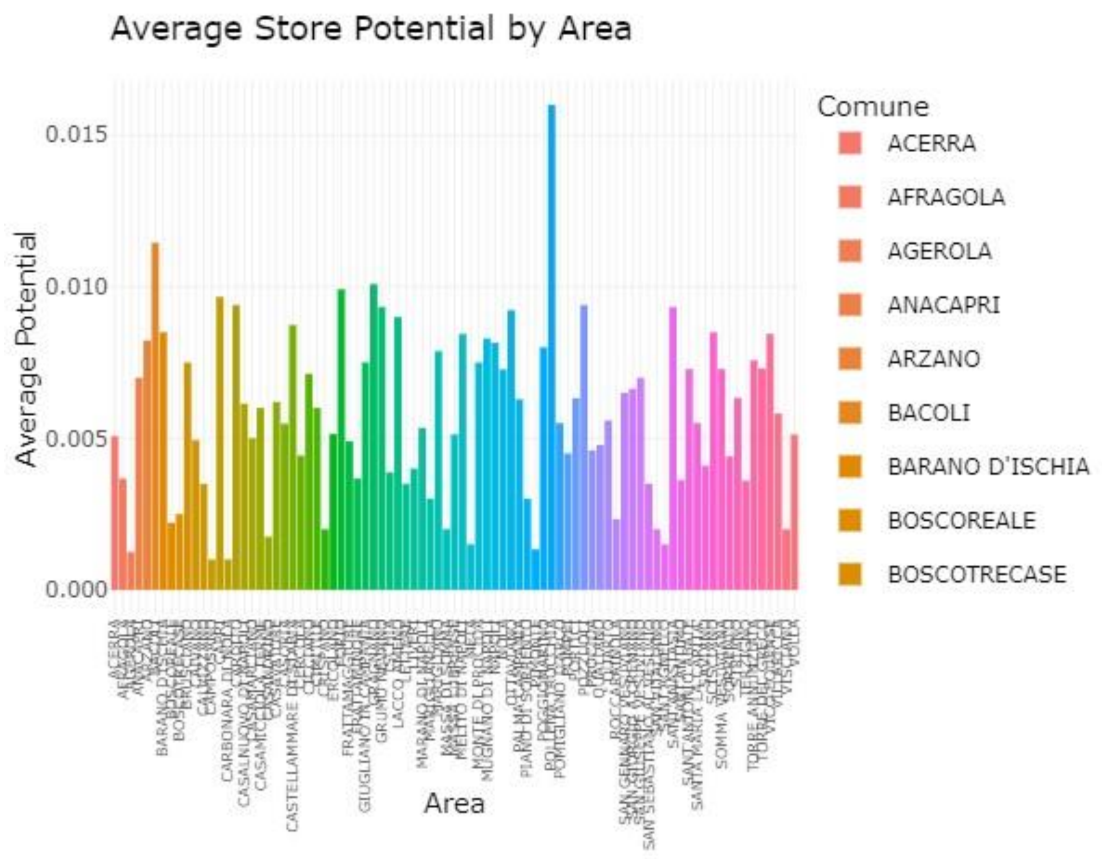
Average Store potential by Area



~~~
```{r}
average_potential <- merged_data %>%
  group_by(Comune) %>%
  summarise(AveragePotential = mean(Potenziale))

ggplotly(ggplot(average_potential, aes(x = Comune, y = AveragePotential, fill = Comune)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1, size=6)) +
  labs(title = "Average Store Potential by Category",
       x = "Category",
       y = "Average Potential"))
~~~

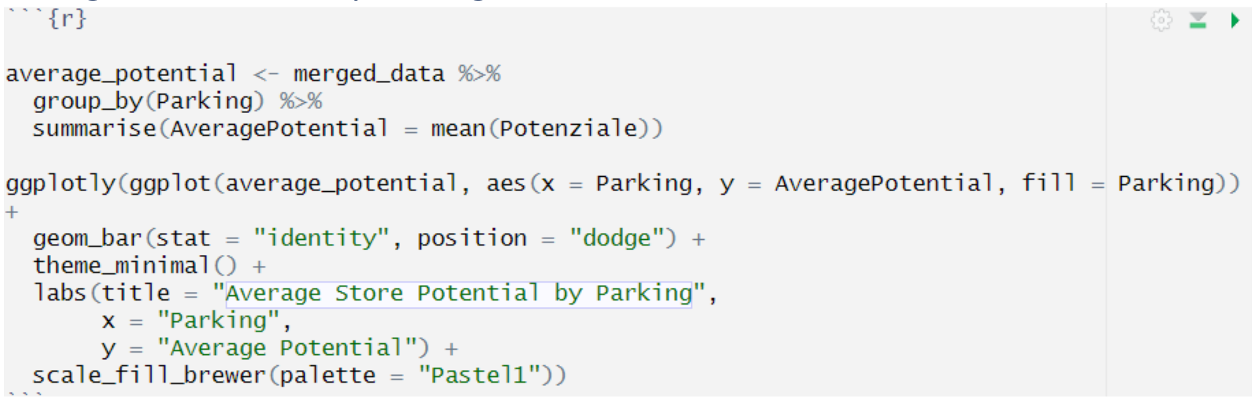
This code calculates the average potential (Potenziale) for each area (Comune) in the merged_data dataframe and then creates an interactive bar plot using ggplotly().

Output:



From the graph above we can conclude that GRAGNANO is the area with highest potential

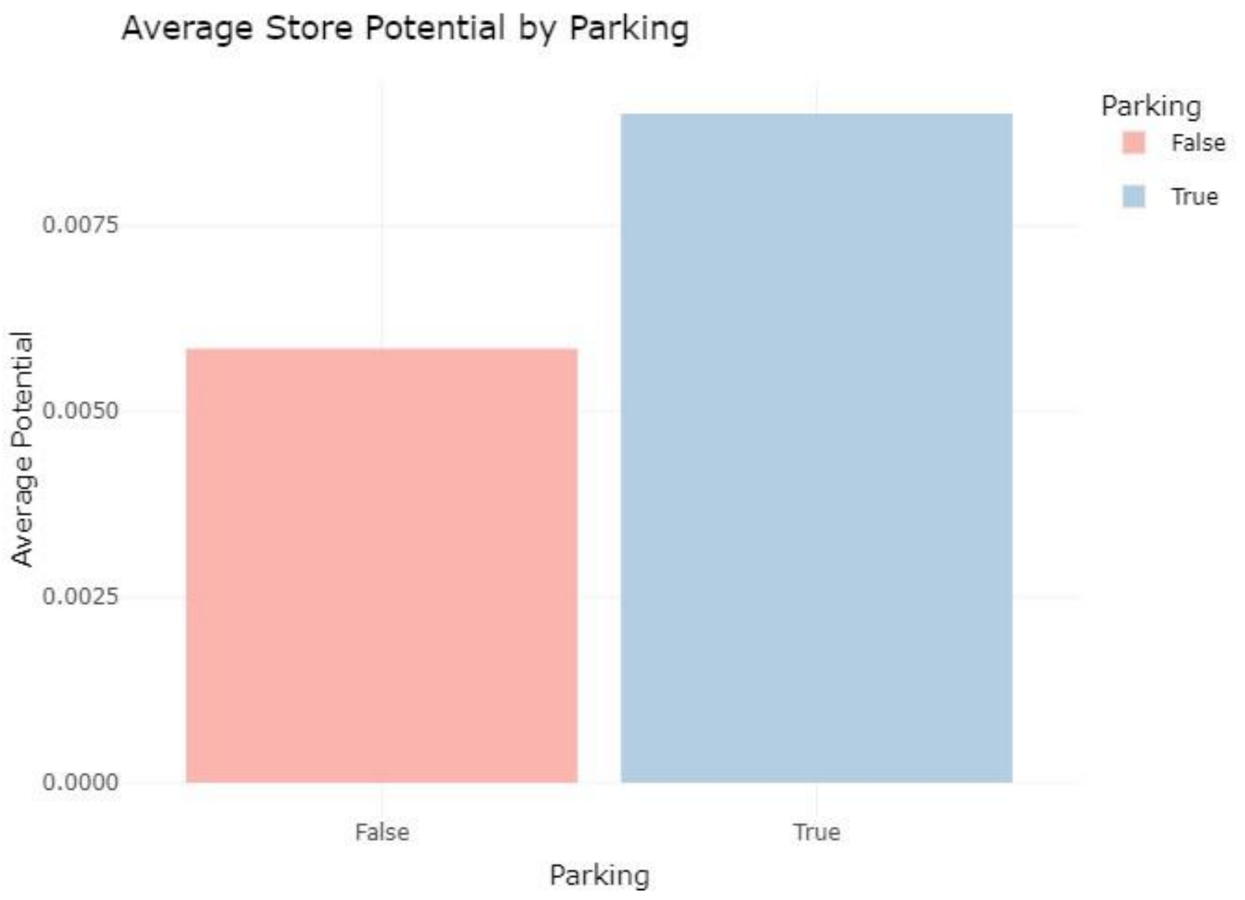
Average Store Potential by Parking



~~~~
```{r}

average_potential <- merged_data %>%
  group_by(Parking) %>%
  summarise(AveragePotential = mean(Potenziale))

ggplotly(ggplot(average_potential, aes(x = Parking, y = AveragePotential, fill = Parking))
+
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Average Store Potential by Parking",
       x = "Parking",
       y = "Average Potential") +
  scale_fill_brewer(palette = "Pastel1"))
```
~~~~

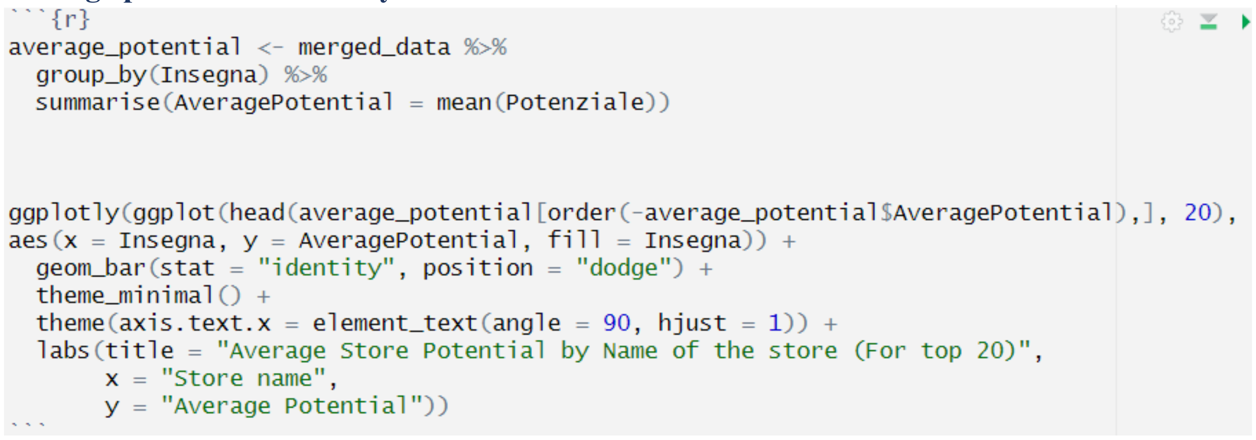
this code creates an interactive bar plot showing the average store potential for each parking category(Parking is present/not-present), providing a visual representation of the differences in potential across different parking options.

Output



As expected, it can be concluded that the stores with Parking have more potential then stores without parking

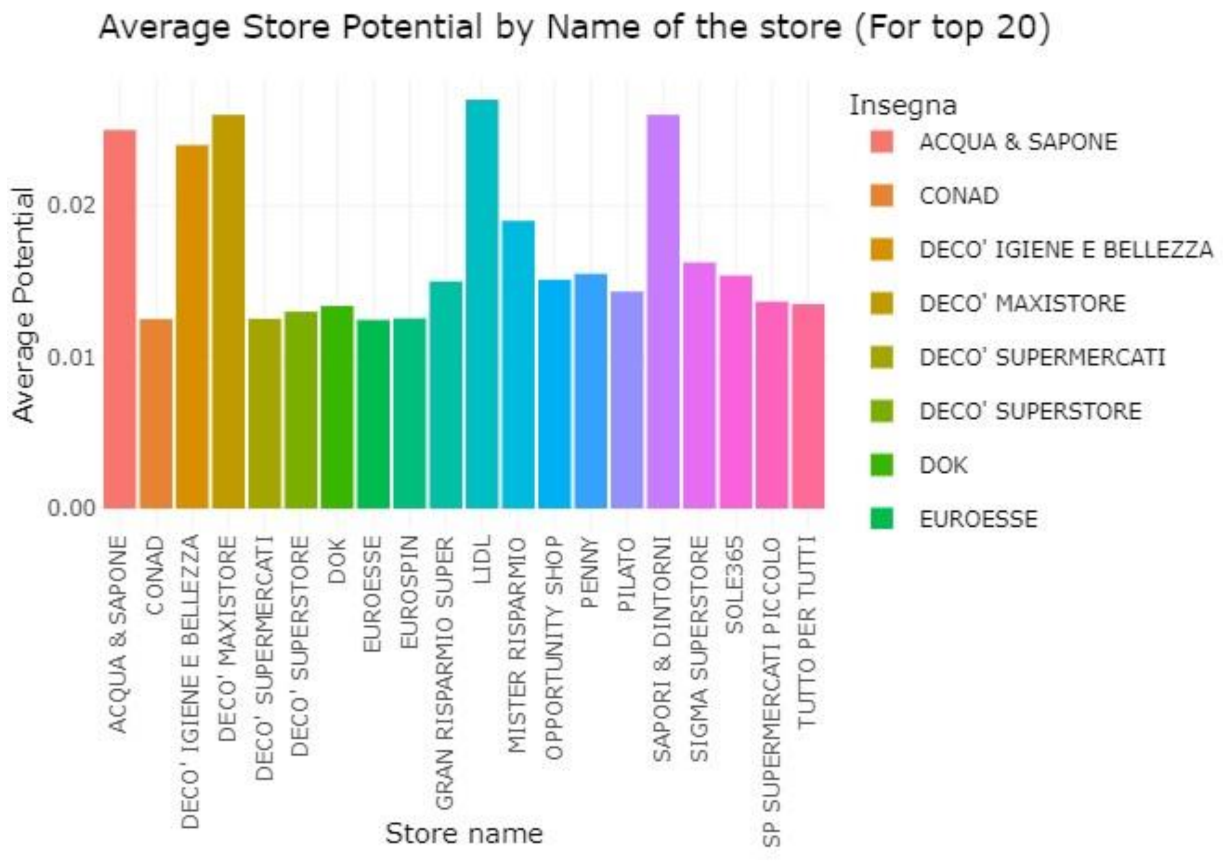
Average potential of store by name of the store



~~~
```{r}
average_potential <- merged_data %>%
  group_by(Insegna) %>%
  summarise(AveragePotential = mean(Potenziale))



ggplotly(ggplot(head(average_potential[order(-average_potential$AveragePotential),], 20),
aes(x = Insegna, y = AveragePotential, fill = Insegna)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "Average Store Potential by Name of the store (For top 20)",
       x = "Store name",
       y = "Average Potential"))
```
~~~

this code creates an interactive bar plot showing the average store potential for the top 10 stores with the highest potential, providing a visual representation of the differences in potential across different stores

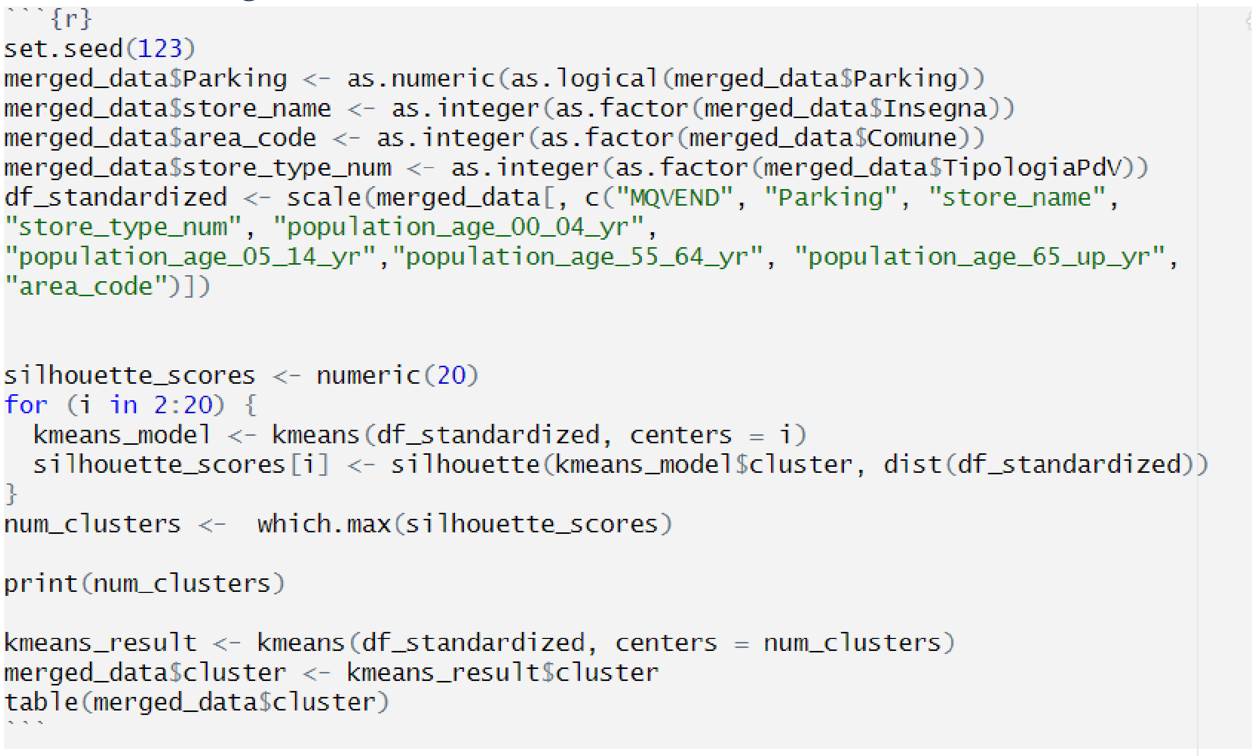
Output:



In conclusion, Mister Risparmio stores make the highest potential overall.

Feature Engineering

K-mean clustering



~~~
```{r}
set.seed(123)
merged_data$Parking <- as.numeric(as.logical(merged_data$Parking))
merged_data$store_name <- as.integer(as.factor(merged_data$Insegna))
merged_data$area_code <- as.integer(as.factor(merged_data$Comune))
merged_data$store_type_num <- as.integer(as.factor(merged_data$TipologiaPdV))
df_standardized <- scale(merged_data[, c("MQVEND", "Parking", "store_name",
"store_type_num", "population_age_00_04_yr",
"population_age_05_14_yr","population_age_55_64_yr", "population_age_65_up_yr",
"area_code")])


silhouette_scores <- numeric(20)
for (i in 2:20) {
  kmeans_model <- kmeans(df_standardized, centers = i)
  silhouette_scores[i] <- silhouette(kmeans_model$cluster, dist(df_standardized))
}
num_clusters <-  which.max(silhouette_scores)

print(num_clusters)

kmeans_result <- kmeans(df_standardized, centers = num_clusters)
merged_data$cluster <- kmeans_result$cluster
table(merged_data$cluster)
```
~~~

This code segment is conducting K-means clustering on the standardized features of the merged_data dataframe. Here's a step-by-step explanation:

Setting Seed and Data Preprocessing:

set.seed(123): Sets the seed for reproducibility.

Data preprocessing steps follow:

merged_data$Parking <- as.numeric(as.logical(merged_data$Parking)): Converts the Parking variable to numeric format after first converting it to logical.

merged_data$store_name <- as.integer(as.factor(merged_data$Insegna)): Converts the Insegna variable to a factor and then to an integer, likely for categorical encoding.

merged_data$area_code <- as.integer(as.factor(merged_data$Comune)): Converts the Comune variable to a factor and then to an integer, likely for categorical encoding.

merged_data$store_type_num <- as.integer(as.factor(merged_data$TipologiaPdV)): Converts the TipologiaPdV variable to a factor and then to an integer, likely for categorical encoding.

df_standardized <- scale(merged_data[, c("MQVEND", "Parking", "store_name", "store_type_num", "population_age_00_04_yr", "population_age_05_14_yr","population_age_55_64_yr", "population_age_65_up_yr", "area_code")]): Standardizes the specified features in the merged_data dataframe and stores them in df_standardized.

Determining Optimal Number of Clusters:

silhouette_scores <- numeric(20): Initializes an empty vector to store silhouette scores.

A loop iterates from 2 to 20 clusters, fitting K-means models and computing silhouette scores for each.

The number of clusters with the highest silhouette score is stored in num_clusters.

Performing K-means Clustering:

kmeans_result <- kmeans(df_standardized, centers = num_clusters): Fits a K-means model with the optimal number of clusters (num_clusters) determined earlier.

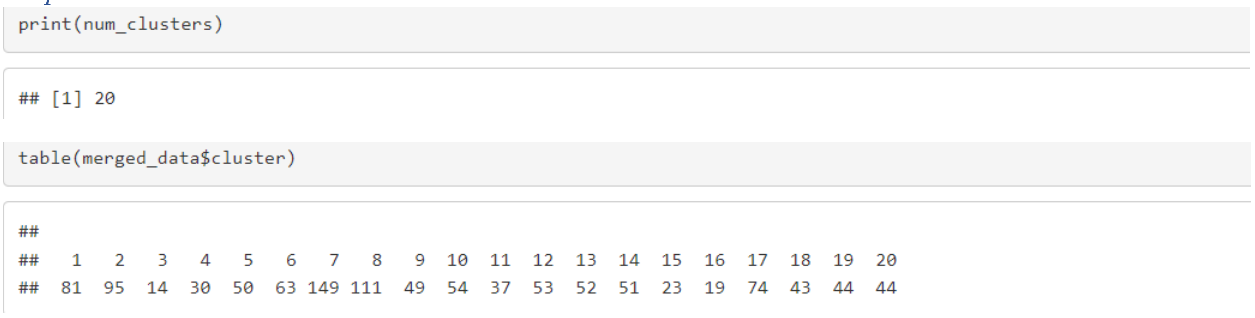
The cluster assignments are stored in merged_data$cluster.

Displaying Cluster Counts:

table(merged_data$cluster): Generates a table displaying the count of observations in each cluster.

this code conducts K-means clustering to identify natural groupings in the data based on standardized features. It then assigns each observation to a cluster and provides a summary of the cluster counts.

Outputs:



```
print(num_clusters)
```

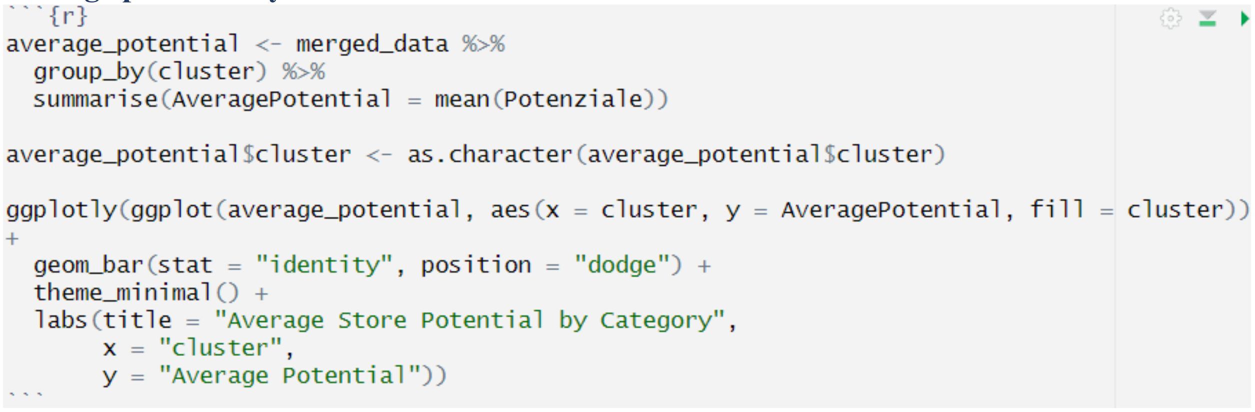
```
## [1] 20
```

```
table(merged_data$cluster)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##  81  95  14  30  50  63 149 111  49  54  37  53  52  51  23  19  74  43  44  44
```

As figured out in above analysis most of the columns does not have coorelation with potential column, this cluster column is then used for random forest classification.

Average potential by cluster



~~~
```{r}
average_potential <- merged_data %>%
  group_by(cluster) %>%
  summarise(AveragePotential = mean(Potenziale))

average_potential$cluster <- as.character(average_potential$cluster)

ggplotly(ggplot(average_potential, aes(x = cluster, y = AveragePotential, fill = cluster))
+
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Average Store Potential by Category",
       x = "cluster",
       y = "Average Potential"))
```
~~~

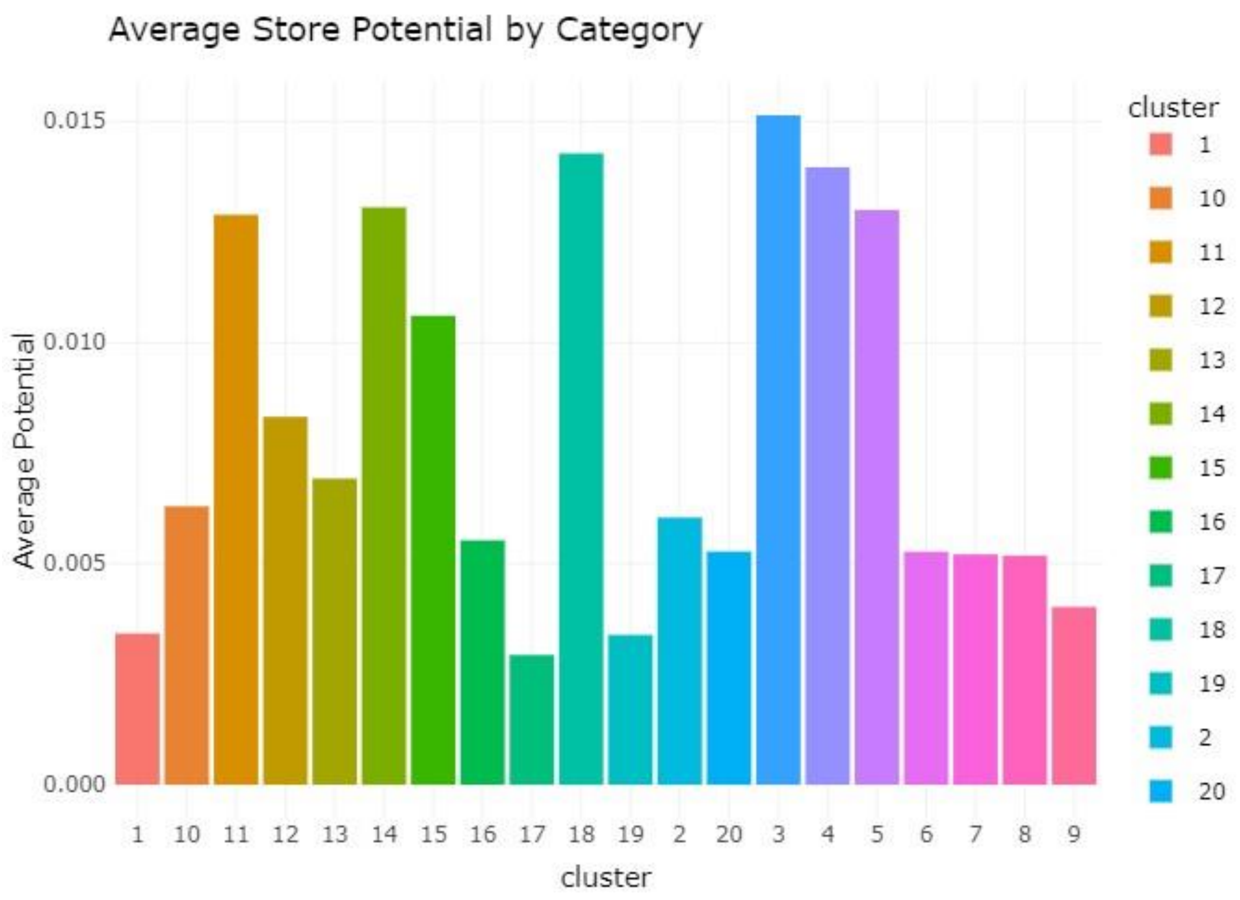
this code produces an interactive bar plot that allows users to explore the average store potential for each cluster visually.

for this we computed the average store potential (Potenziale) for each cluster by first grouping the data by the cluster variable and then calculating the mean of Potenziale within each group. The results are stored in a new dataframe called average_potential.

then converted the cluster column in the average_potential dataframe to a character type. Converting it to a character type is often useful when dealing with categorical variables to ensure correct plotting.

it provides insights into the average population distribution across different clusters identified by K-means clustering, allowing for visual exploration and analysis of population patterns.

Output:



The output above concludes that each cluster have different potential hence variable clusters can be used as a distinguisher between potential in random forest tree

Cluster Count

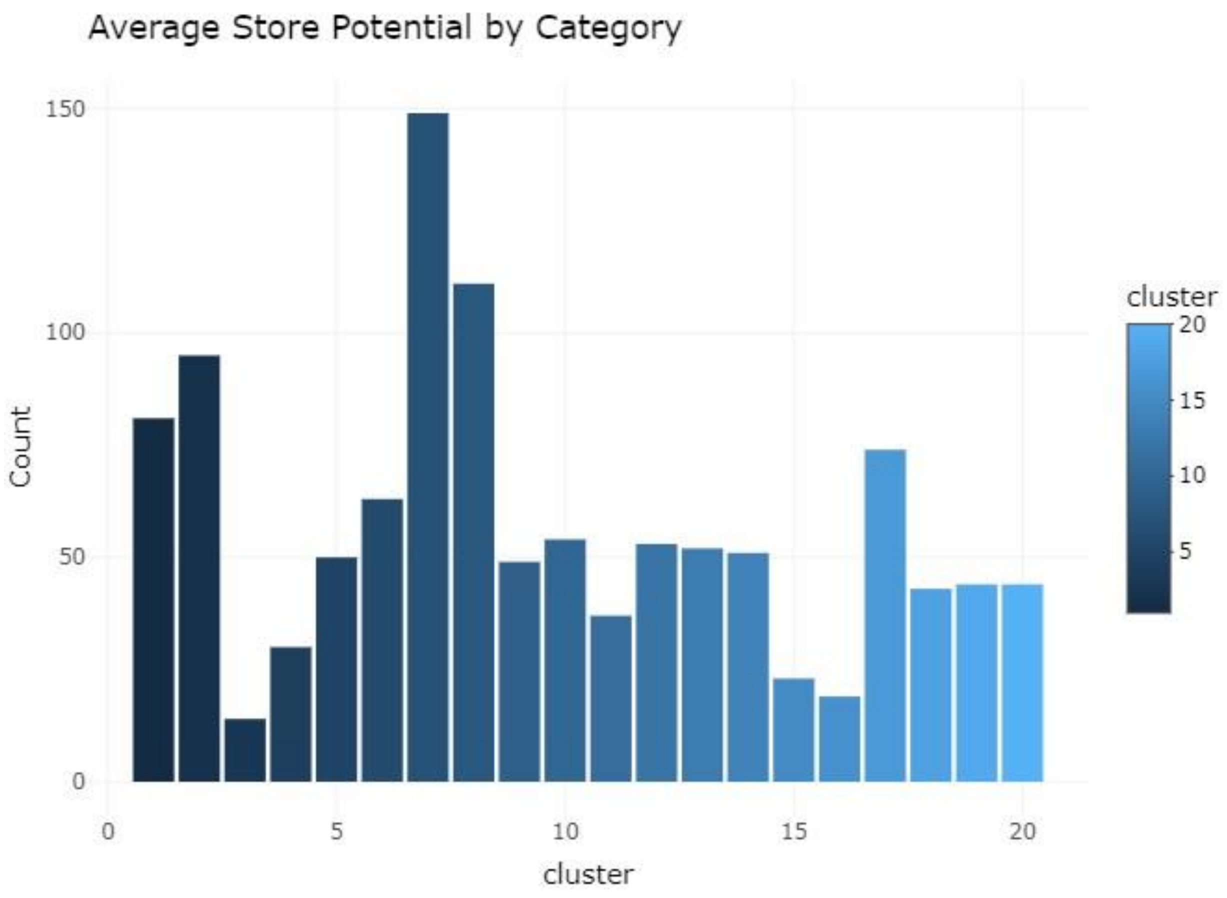
~~~
```{r}
cluster_count <- merged_data %>%
  group_by(cluster) %>%
  count()
#average_potential$cluster <- as.character(average_potential$cluster)
ggplotly(ggplot(cluster_count, aes(x = cluster, y = n, fill = cluster)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Average Store Potential by Category",
       x = "cluster",
       y = "Count"))

```
~~~

This code snippet calculates the count of observations within each cluster in the merged_data dataframe and creates an interactive bar plot to visualize the distribution of observations across different clusters.

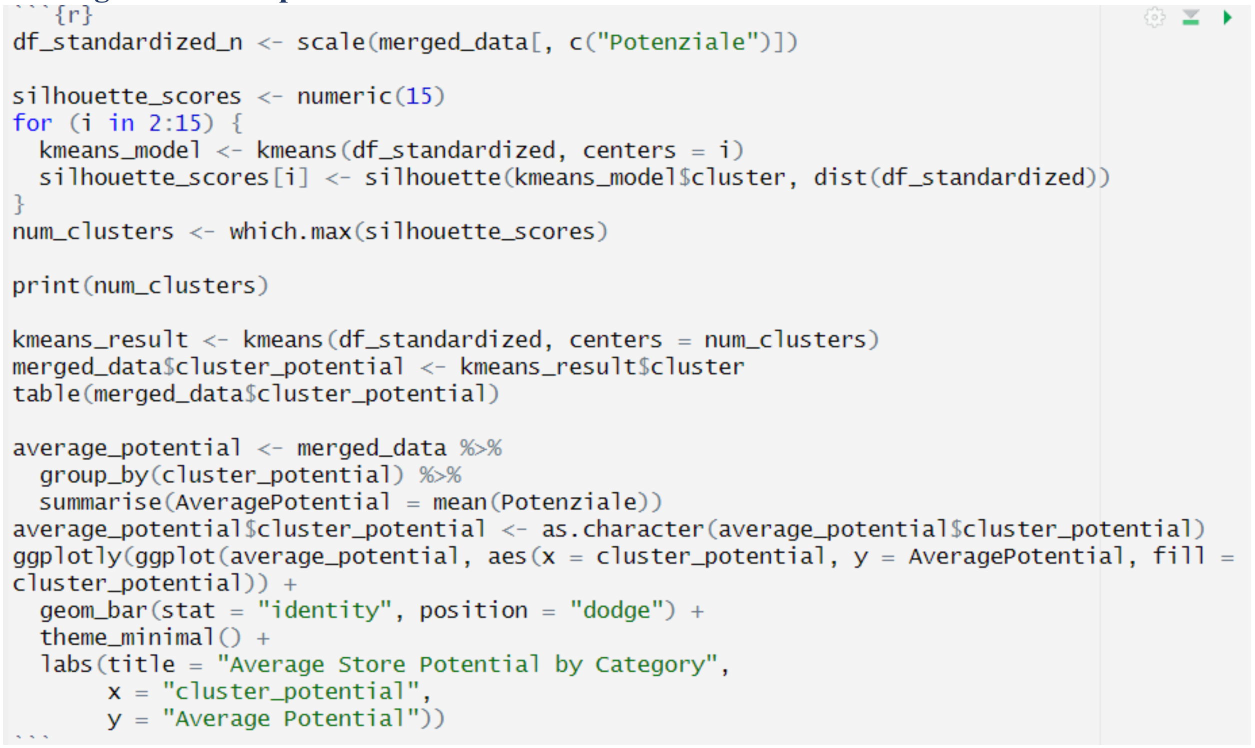
By visualizing the distribution of observations across clusters, we can gain insights into the distribution and density of data points within each cluster.

This type of plot is useful for understanding the composition of clusters and identifying any imbalances or biases in the clustering process.

Output:



Making clusters for potential



~~~
```{r}
df_standardized_n <- scale(merged_data[, c("Potenziale")])

silhouette_scores <- numeric(15)
for (i in 2:15) {
  kmeans_model <- kmeans(df_standardized, centers = i)
  silhouette_scores[i] <- silhouette(kmeans_model$cluster, dist(df_standardized))
}
num_clusters <- which.max(silhouette_scores)

print(num_clusters)

kmeans_result <- kmeans(df_standardized, centers = num_clusters)
merged_data$cluster_potential <- kmeans_result$cluster
table(merged_data$cluster_potential)

average_potential <- merged_data %>%
  group_by(cluster_potential) %>%
  summarise(AveragePotential = mean(Potenziale))
average_potential$cluster_potential <- as.character(average_potential$cluster_potential)
ggplotly(ggplot(average_potential, aes(x = cluster_potential, y = AveragePotential, fill =
cluster_potential)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  labs(title = "Average Store Potential by Category",
       x = "cluster_potential",
       y = "Average Potential"))
```
~~~

performs k-means clustering on the standardized Potenziale (Potential) variable in the merged_data dataframe to group stores based on their potential

Standardize Potenziale Variable:

df_standardized <- scale(merged_data[, c("Potenziale")]): This standardizes the Potenziale variable by subtracting the mean and dividing by the standard deviation.

Determine Optimal Number of Clusters:

It computes the silhouette scores for different numbers of clusters (ranging from 2 to 15) to find the optimal number of clusters.

For each number of clusters, it performs k-means clustering and calculates the silhouette score.

The number of clusters with the highest silhouette score is chosen as the optimal number of clusters.

Perform K-Means Clustering:

kmeans_result <- kmeans(df_standardized, centers = num_clusters): This performs k-means clustering on the standardized Potenziale variable with the optimal number of clusters determined earlier.

Assign Cluster Labels:

merged_data$cluster_potential <- kmeans_result$cluster: This assigns cluster labels to each store in the merged_data dataframe based on the clustering results.

Summarize Clusters:

It summarizes the average potential for each cluster.

average_potential <- merged_data %>% group_by(cluster_potential) %>% summarise(AveragePotential = mean(Potenziale)): This computes the average potential for each cluster by taking the mean of Potenziale within each cluster.

Visualize Cluster Analysis:

It creates an interactive bar plot using plotly to visualize the average store potential by cluster.

ggplotly() converts the ggplot bar plot into an interactive plotly plot.

The plot displays the average potential on the y-axis and the cluster labels on the x-axis.

Overall, we clustered stores based on their potential and visualizes the average potential of each cluster, providing insights into different groups of stores based on their potential.

Output:

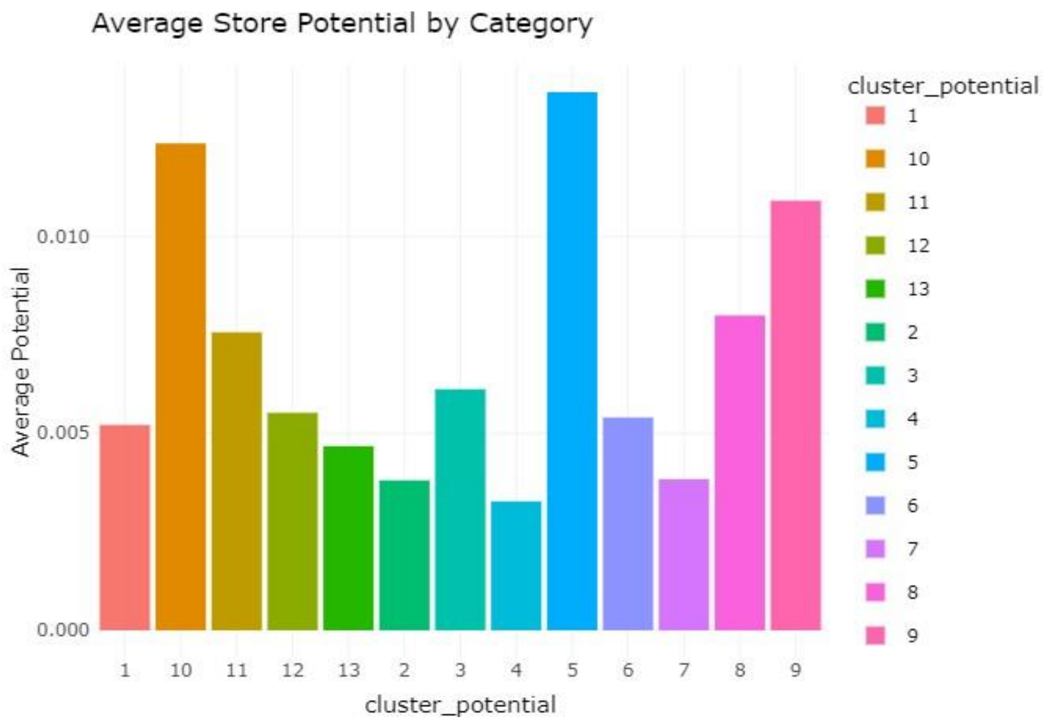
```
print(num_clusters)
```

```
## [1] 13
```

```
table(merged_data$cluster_potential)
```

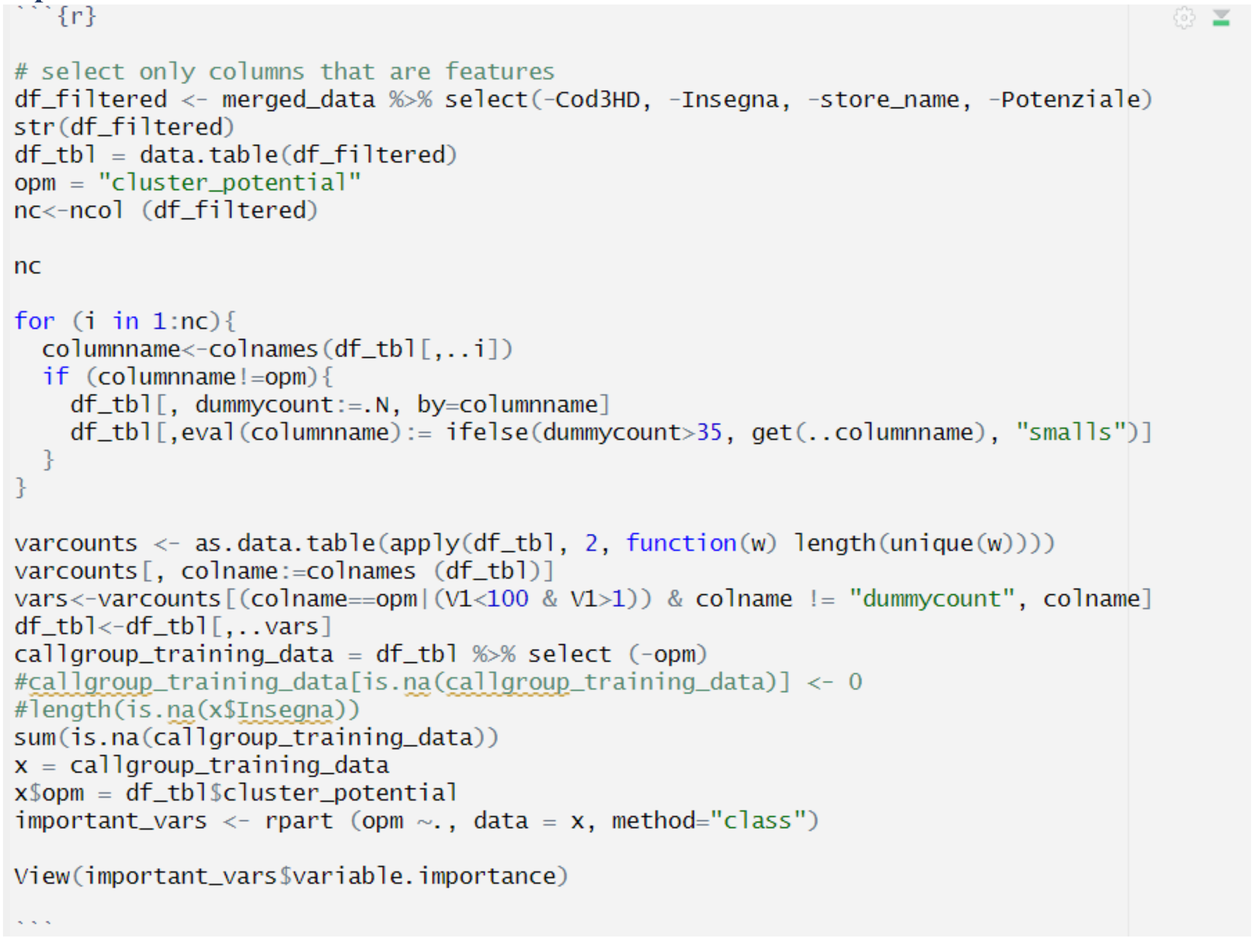
```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13
## 147  68  96 123 110  54  99  70  83  86  63  19 118
```

This is an attempt to create categories of target variable to be able to run random forest classifier



Feature Importance

R part



~~~
```{r}

# select only columns that are features
df_filtered <- merged_data %>% select(-Cod3HD, -Insegna, -store_name, -Potenziale)
str(df_filtered)
df_tbl = data.table(df_filtered)
opm = "cluster_potential"
nc<-ncol (df_filtered)

nc

for (i in 1:nc){
  columnname<-colnames(df_tbl[,..i])
  if (columnname!=opm){
    df_tbl[, dummycount:=.N, by=columnname]
    df_tbl[,eval(columnname):= ifelse(dummycount>35, get(..columnname), "smalls")]
  }
}

varcounts <- as.data.table(apply(df_tbl, 2, function(w) length(unique(w))))
varcounts[, colname:=colnames (df_tbl)]
vars<-varcounts[(colname==opm|(V1<100 & V1>1)) & colname != "dummycount", colname]
df_tbl<-df_tbl[,..vars]
callgroup_training_data = df_tbl %>% select (-opm)
#callgroup_training_data[is.na(callgroup_training_data)] <- 0
#length(is.na(x$Insegna))
sum(is.na(callgroup_training_data))
x = callgroup_training_data
x$opm = df_tbl$cluster_potential
important_vars <- rpart (opm ~., data = x, method="class")

View(important_vars$variable.importance)

```
~~~

This code snippet appears to be a part of a data preprocessing and feature selection procedure.

Select Only Feature Columns:

for this firstly, we selected only the columns that are considered features (excluding columns Cod3HD, Insegna, and store_name) from the merged_data dataframe.

Convert Data to Data Table:

Then converted the df_filtered dataframe to a data table format using the data.table library.

Define Target Variable:

opm = " cluster_potential": This assigns the name of the target variable (the variable to be predicted) to the variable opm. In this case, the target variable is named " cluster_potential".

Loop Through Columns for Feature Engineering:

This loop iterates over each column in the dataframe to perform feature engineering.

columnname<-colnames(df_tbl[,..i]): This line retrieves the name of the current column being processed.

Inside the loop:

The number of unique values in each column is counted and stored in dummycount using df_tbl[, dummycount:=.N, by=columnname].If the current column is not the target variable (opm), and the count of unique values (dummycount) is greater than 80, the column values are kept; otherwise, they are replaced with "smalls".This process effectively removes columns with too few unique values, potentially eliminating irrelevant or noisy features.

Filter Variables for Model Building:

varcounts <- as.data.table(apply(df_tbl, 2, function(w) length(unique(w)))): This calculates the count of unique values for each column in the dataframe. vars<-varcounts[(colname==opm|(V1<100 & V1>1)) & colname != "dummycount", colname]: This selects columns where the count of unique values is between 2 and 100, or where the column is the target variable (opm).

Prepare Data for Modeling:

callgroup_training_data = df_tbl %>% select (-opm): This selects all columns except the target variable (opm) for model training. Missing values in the dataset are checked and handled (though this part is commented out).

Build a Classification Model:

important_vars <- rpart (opm ~., data = x, method="class"): This builds a classification model using the rpart algorithm (method="class") with the target variable (opm) predicted by all other variables in the dataset.

View Variable Importance:

View(important_vars$variable.importance): This displays the variable importance scores calculated by the classification model, indicating the importance of each feature in predicting the target variable.

Overall, twe performed feature engineering, filters variables for model building, prepares the data for modeling, builds a classification model, and examines the importance of features in predicting the target variable.

Output:

```
## 'data.frame':    1136 obs. of  23 variables:
##  $ TipologiaPdV           : chr  "LIS" "LIS" "SUP" "LIS" ...
##  $ MQVEND                 : int  250 300 800 120 400 200 150 1400 500 400 ...
##  $ Parking                : num  0 1 1 0 0 0 0 1 1 0 ...
##  $ Indirizzo              : chr  "VIA GIOVANNI DE FRAJA 32/40" "VIA CUCCARO 1" "VIA CINQUE VIE 45/47" "VIA LIBERTINI 6"
...
##  $ Comune                 : chr  "POZZUOLI" "QUARTO" "AFRAGOLA" "CAIVANO" ...
##  $ microcode              : num  6.31e+11 6.31e+11 6.30e+11 6.30e+11 6.30e+11 ...
##  $ district               : chr  "POZZUOLI" "QUARTO" "AFRAGOLA" "CAIVANO" ...
##  $ population             : num  117 140 260 492 293 ...
##  $ population_m           : int  54 64 133 249 139 485 199 43 83 465 ...
##  $ population_f           : int  63 76 127 243 154 505 229 56 87 542 ...
##  $ population_age_00_04_yr: int  4 6 11 19 11 32 14 7 9 38 ...
##  $ population_age_05_14_yr: int  15 15 43 63 29 103 39 9 18 104 ...
##  $ population_age_15_34_yr: int  23 28 74 135 68 218 108 26 23 230 ...
##  $ population_age_35_44_yr: int  22 21 55 63 42 157 56 17 30 140 ...
##  $ population_age_45_54_yr: int  25 17 33 78 29 173 72 15 15 118 ...
##  $ population_age_55_64_yr: int  16 15 19 67 50 113 58 11 25 161 ...
##  $ population_age_65_up_yr: int  12 38 26 66 63 194 81 15 50 217 ...
##  $ TipologiaPdV_cat       : Factor w/ 4 levels "1","2","3","4": 2 2 4 2 4 2 2 1 4 4 ...
##  $ Parking_cat            : Factor w/ 2 levels "1","2": 1 2 2 1 1 1 1 2 2 1 ...
##  $ area_code              : int  59 62 2 11 22 26 41 49 49 56 ...
##  $ store_type_num         : int  2 2 4 2 4 2 2 1 4 4 ...
##  $ cluster                : int  8 2 4 17 11 6 7 12 18 5 ...
##  $ cluster_potential      : int  1 1 5 7 9 3 4 11 5 10 ...
##  - attr(*, "na.action")= 'omit' Named int [1:14] 211 302 307 315 364 387 688 712 812 921 ...
##   ..- attr(*, "names")= chr [1:14] "211" "302" "307" "315" ...
```

```
nc
```

```
## [1] 23
```

```
sum(is.na(callgroup_training_data))
```

```
## [1] 0
```

The final output concludes that store_type and cluster are important features for creating decision tree. Hence, they are used for building random forest tree.

Hypothesis testing

~~~
```{r}
result <- chisq.test(merged_data$cluster_potential, merged_data$cluster)
print(result)

result <- chisq.test(merged_data$cluster_potential, merged_data$store_type_num)
print(result)

result <- chisq.test(merged_data$cluster_potential, merged_data$area_code)
print(result)
```
~~~

Chi-squared Test between cluster_potential and cluster:

This code conducts a chi-squared test to assess the independence between cluster_potential and cluster. The chisq.test() function calculates the test statistic, p-value, and other relevant statistics.

The result is then printed to the console.

Chi-squared Test between cluster_potential and store_type_num:

Similar to the first test, this code segment conducts a chi-squared test to assess the independence between cluster_potential and store_type_num, which is a categorical variable representing store types.

The result of the test is printed to the console.

Chi-squared Test between cluster_potential and area_code:

This part of the code performs a chi-squared test to assess the independence between cluster_potential and area_code, which is a numeric variable representing area codes.

The result of the test is printed to the console.

These tests help determine whether there is a significant association between cluster_potential and the other variables. If the p-value is below a certain significance level (e.g., 0.05), it suggests that there is evidence to reject the null hypothesis of independence, indicating that there may be a significant association between the variables.

Output:

```
result <- chisq.test(merged_data$cluster_potential, merged_data$cluster)
print(result)
```

```
##
##  Pearson's Chi-squared test
##
## data:  merged_data$cluster_potential and merged_data$cluster
## X-squared = 9056.1, df = 228, p-value < 2.2e-16
```

```
result <- chisq.test(merged_data$cluster_potential, merged_data$store_type_num)
print(result)
```

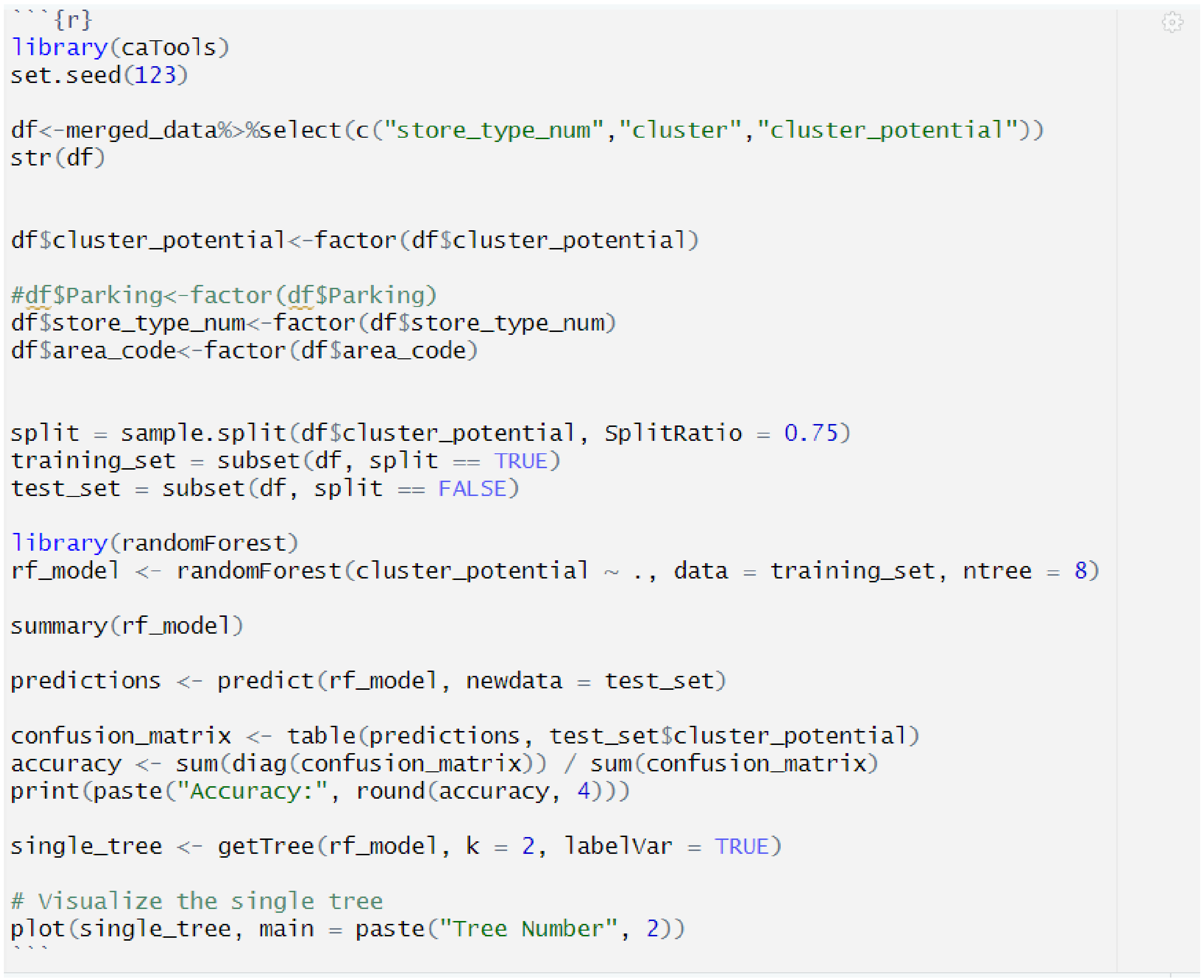
```
##
##  Pearson's Chi-squared test
##
## data:  merged_data$cluster_potential and merged_data$store_type_num
## X-squared = 1979.6, df = 36, p-value < 2.2e-16
```

```
result <- chisq.test(merged_data$cluster_potential, merged_data$area_code)
print(result)
```

```
##
##  Pearson's Chi-squared test
##
## data:  merged_data$cluster_potential and merged_data$area_code
## X-squared = 2357.9, df = 1008, p-value < 2.2e-16
```

Modeling

Random Forest model



~~~r
```{r}
library(caTools)
set.seed(123)

df<-merged_data%>%select(c("store_type_num","cluster","cluster_potential"))
str(df)


df$cluster_potential<-factor(df$cluster_potential)

#df$Parking<-factor(df$Parking)
df$store_type_num<-factor(df$store_type_num)
df$area_code<-factor(df$area_code)


split = sample.split(df$cluster_potential, SplitRatio = 0.75)
training_set = subset(df, split == TRUE)
test_set = subset(df, split == FALSE)

library(randomForest)
rf_model <- randomForest(cluster_potential ~ ., data = training_set, ntree = 8)

summary(rf_model)

predictions <- predict(rf_model, newdata = test_set)

confusion_matrix <- table(predictions, test_set$cluster_potential)
accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
print(paste("Accuracy:", round(accuracy, 4)))

single_tree <- getTree(rf_model, k = 2, labelVar = TRUE)

# Visualize the single tree
plot(single_tree, main = paste("Tree Number", 2))
~~~
```

Data Preprocessing:

we Select columns "store_type_num", "cluster", and "cluster_potential" from the merged_data dataframe.Convert categorical variables to factors: "store_type_num", "cluster", and "cluster_potential".

Split the dataset into training and testing sets using sample.split() from the caTools library.

Model Training:

Import the randomForest library.Build a random forest model (rf_model) to predict the "cluster_potential" variable based on the selected features ("store_type_num" and "cluster").The ntree parameter specifies the number of trees in the forest.

Model Evaluation:

Generate a summary of the random forest model using summary(rf_model).Make predictions on the test set using the trained random forest model (predictions <- predict(rf_model, newdata = test_set)).Construct
```

a confusion matrix to evaluate the performance of the model.Calculate the accuracy of the model based on the confusion matrix.Print the accuracy score.

**Visualizing a Single Tree:**

Extract a single tree from the random forest model using getTree() with k = 2 (indicating the second tree).Plot the single tree using plot().

Overall, we train a random forest model to predict the "cluster_potential" variable based on "store_type_num" and "cluster" features, evaluates its performance, and visualizes one of the trees in the forest.

***Output:***

```
str(df)
```

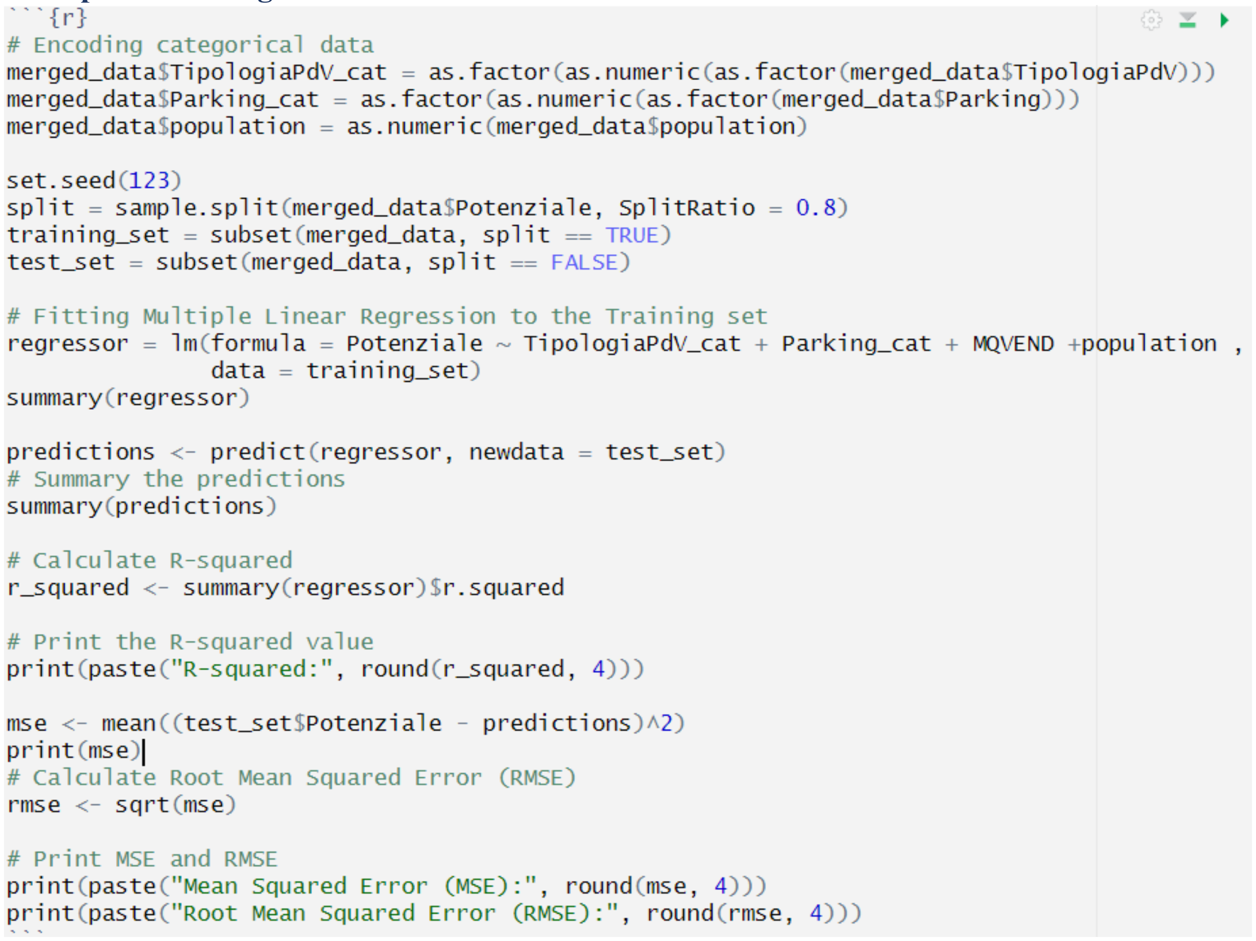
```
## 'data.frame':    1136 obs. of  3 variables:
##  $ store_type_num   : int  2 2 4 2 4 2 2 1 4 4 ...
##  $ cluster          : int  8 2 4 17 11 6 7 12 18 5 ...
##  $ cluster_potential: int  1 1 5 7 9 3 4 11 5 10 ...
##  - attr(*, "na.action")= 'omit' Named int [1:14] 211 302 307 315 364 387 688 712 812 921 ...
##   ..- attr(*, "names")= chr [1:14] "211" "302" "307" "315" ...
```

```
print(paste("Accuracy:", round(accuracy * 100, 2),'%'))
```

```
## [1] "Accuracy: 72.92 %"
```

The accuracy above shows that the model has good capability to predict the cluster of potential with given type of store and cluster which was made earlier using many different columns.

Multiple Linear regression:



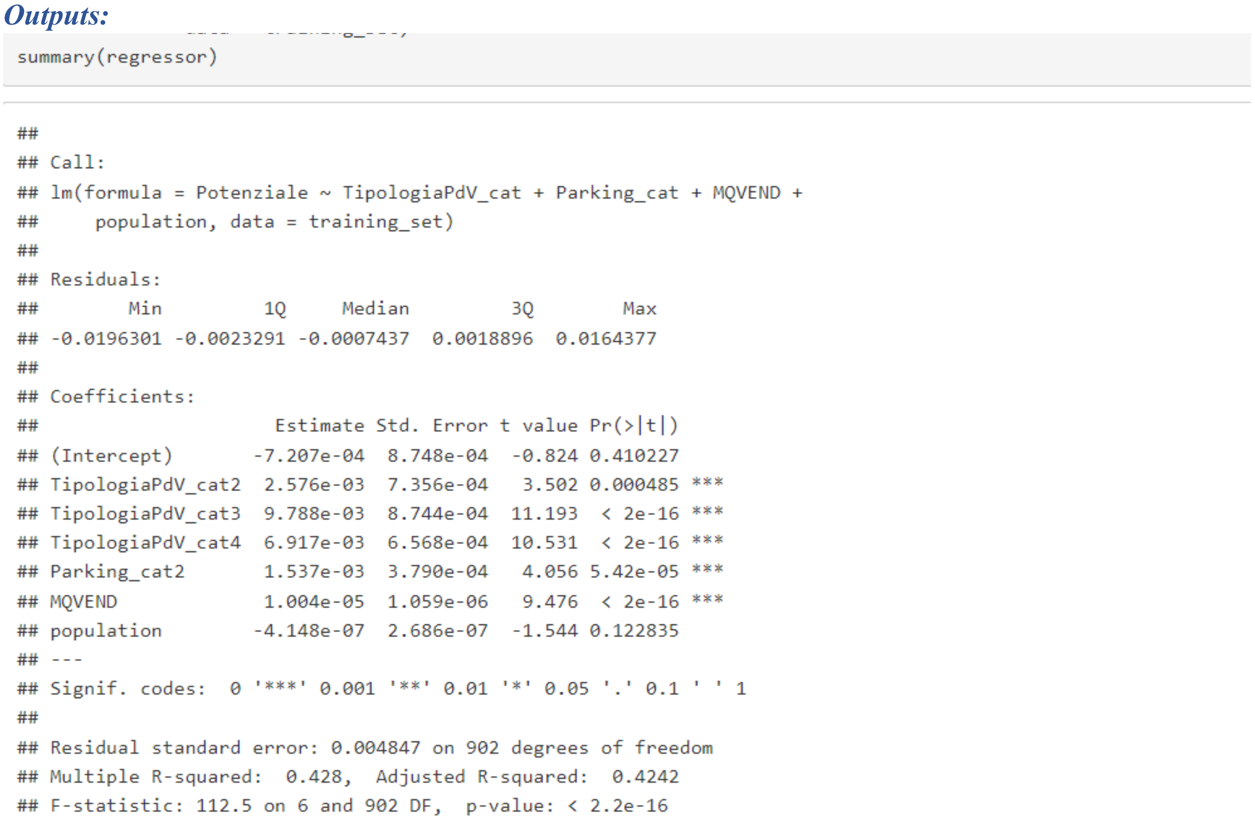
~~~r
```{r}
# Encoding categorical data
merged_data$TipologiaPdV_cat = as.factor(as.numeric(as.factor(merged_data$TipologiaPdV)))
merged_data$Parking_cat = as.factor(as.numeric(as.factor(merged_data$Parking)))
merged_data$population = as.numeric(merged_data$population)

set.seed(123)
split = sample.split(merged_data$Potenziale, SplitRatio = 0.8)
training_set = subset(merged_data, split == TRUE)
test_set = subset(merged_data, split == FALSE)

# Fitting Multiple Linear Regression to the Training set
regressor = lm(formula = Potenziale ~ TipologiaPdV_cat + Parking_cat + MQVEND +population ,
               data = training_set)
summary(regressor)

predictions <- predict(regressor, newdata = test_set)
# Summary the predictions
summary(predictions)

# Calculate R-squared
r_squared <- summary(regressor)$r.squared

# Print the R-squared value
print(paste("R-squared:", round(r_squared, 4)))

mse <- mean((test_set$Potenziale - predictions)^2)
print(mse)
# Calculate Root Mean Squared Error (RMSE)
rmse <- sqrt(mse)

# Print MSE and RMSE
print(paste("Mean Squared Error (MSE):", round(mse, 4)))
print(paste("Root Mean Squared Error (RMSE):", round(rmse, 4)))
~~~
```

**Data Preprocessing:**

Converts categorical variables "TipologiaPdV" and "Parking" into numeric factors.Converts the "population" variable to numeric.

**Data Splitting:**

Splits the dataset into training and testing sets using sample.split() from the caTools library.

**Model Training:**

Fit a multiple linear regression model (lm) to the training set with "Potenziale" as the dependent variable and "TipologiaPdV_cat", "Parking_cat", "MQVEND", and "population" as independent variables.

**Model Evaluation:**

Summarize the fitted regression model using summary(regressor). Make predictions on the test set using the trained regression model. Calculate the mean squared error (MSE) between the actual "Potenziale" values in the test set and the predicted values. Calculate the root mean squared error (RMSE) from the MSE.
```

Result Presentation:

Print the R-squared value to assess the goodness-of-fit of the model. Print the MSE and RMSE to evaluate the accuracy of the model.

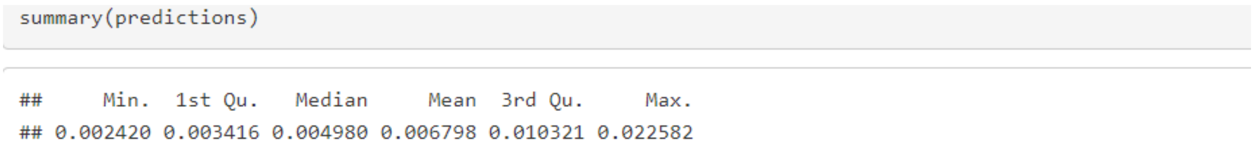
Overall, we preprocessed the data, train a multiple linear regression model, evaluates its performance, and presents the results including R-squared, MSE, and RMSE.

Outputs:



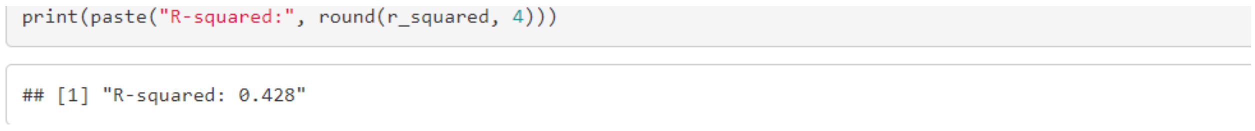
```
summary(regressor)
```

```
##
## Call:
## lm(formula = Potenziale ~ TipologiaPdV_cat + Parking_cat + MQVEND +
##     population, data = training_set)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -0.0196301 -0.0023291 -0.0007437  0.0018896  0.0164377
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -7.207e-04  8.748e-04  -0.824 0.410227
## TipologiaPdV_cat2  2.576e-03  7.356e-04   3.502 0.000485 ***
## TipologiaPdV_cat3  9.788e-03  8.744e-04  11.193  < 2e-16 ***
## TipologiaPdV_cat4  6.917e-03  6.568e-04  10.531  < 2e-16 ***
## Parking_cat2       1.537e-03  3.790e-04   4.056 5.42e-05 ***
## MQVEND             1.004e-05  1.059e-06   9.476  < 2e-16 ***
## population        -4.148e-07  2.686e-07  -1.544 0.122835
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.004847 on 902 degrees of freedom
## Multiple R-squared:  0.428,  Adjusted R-squared:  0.4242
## F-statistic: 112.5 on 6 and 902 DF,  p-value: < 2.2e-16
```



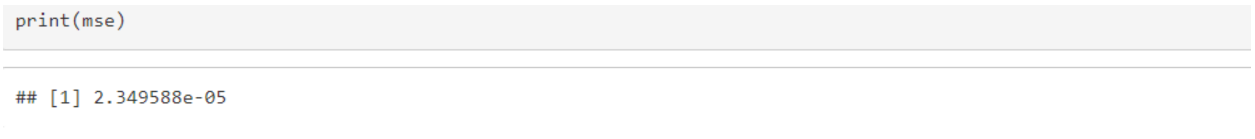
```
summary(predictions)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## 0.002420 0.003416 0.004980 0.006798 0.010321 0.022582
```



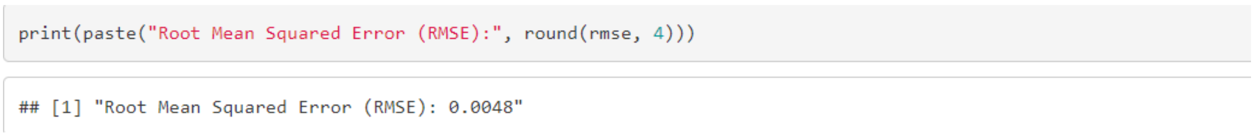
```
print(paste("R-squared:", round(r_squared, 4)))
```

```
## [1] "R-squared: 0.428"
```



```
print(mse)
```

```
## [1] 2.349588e-05
```



```
print(paste("Root Mean Squared Error (RMSE):", round(rmse, 4)))
```

```
## [1] "Root Mean Squared Error (RMSE): 0.0048"
```

Coefficients:

The intercept term is not statistically significant, indicated by a p-value of 0.4102.

The variables TipologiaPdV_cat2, TipologiaPdV_cat3, TipologiaPdV_cat4, Parking_cat2, MQVEND, and population are all statistically significant predictors. Their coefficients are all significantly different from zero ($p < 0.05$).

The coefficient estimates provide insights into the direction and magnitude of the relationships between predictors and the response variable. For example, a one-unit increase in TipologiaPdV_cat2 corresponds to an estimated increase of 0.002576 units in the response variable, holding other variables constant.

Model Fit :

The model explains approximately 42.8% of the variance in the response variable, as indicated by the Multiple R-squared value.

The Adjusted R-squared value is slightly lower, suggesting that the inclusion of additional predictors does not significantly improve the model's explanatory power.

The F-statistic is highly significant ($p < 0.05$), indicating that the model as a whole is significant in predicting the response variable.

The residual standard error provides a measure of the typical deviation of the observed values from the fitted values. In this case, it is approximately 0.004847.

Predictor Importance:

Among the predictors, TipologiaPdV, Parking, and MQVEND appear to be the most influential in predicting the response variable, based on their coefficient estimates and significance levels.

population does not appear to be statistically significant in predicting the response variable, as indicated by its p-value of 0.1228.