

# ***Red Wine Quality***

## ***Analysis using R programming***

*The red wine quality dataset is a comprehensive collection of attributes related to the sensory evaluation of red wine, encompassing various chemical properties and quality ratings. It serves as a valuable resource for understanding the factors influencing the perceived quality of red wine and can aid in predicting or assessing wine quality based on its compositional characteristics.*

*The dataset likely includes features such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol content, among others. These features are indicative of both intrinsic and extrinsic factors affecting wine quality, including grape variety, fermentation processes, and storage conditions.*

*Through statistical analysis and machine learning techniques, researchers can delve into the relationships between these features and wine quality ratings provided by experts or consumers. Insights gained from this analysis can inform winemaking practices, such as adjusting fermentation parameters or optimizing storage conditions, to enhance the overall quality of red wine.*

*Moreover, this dataset can facilitate the development of predictive models to assess wine quality based on objective chemical measurements, potentially saving time and resources in the wine industry's quality control processes. Overall, the red wine quality dataset offers a valuable opportunity for researchers and wine enthusiasts alike to deepen their understanding of red wine composition and quality perception.*

*The red wine quality dataset likely contains a variety of columns, each representing different attributes or characteristics of red wine samples. Here's a general interpretation of some common columns you might find in such a dataset:*

- **Fixed acidity:** This column represents the concentration of non-volatile acids present in the wine, which contribute to its overall acidity level.
- **Volatile acidity:** This column indicates the concentration of volatile acids in the wine, which can affect its aroma and taste, with high levels potentially indicating spoilage.
- **Citric acid:** The citric acid content of the wine contributes to its freshness and acidity, adding to its overall flavor profile.
- **Residual sugar:** This column represents the amount of sugar remaining in the wine after fermentation, influencing its sweetness level.
- **Chlorides:** The concentration of chlorides in the wine can impact its taste and mouthfeel, with higher levels potentially indicating poor quality.
- **Free sulfur dioxide:** This column measures the amount of sulfur dioxide present in its free form, which serves as a preservative and antioxidant in wine.
- **Total sulfur dioxide:** This column represents the total amount of sulfur dioxide present in the wine, including both free and bound forms.
- **Density:** The density of the wine is influenced by its sugar and alcohol content, providing insights into its overall composition.
- **pH:** The pH level of the wine affects its acidity and stability, with lower pH values indicating higher acidity.
- **Sulphates:** This column represents the concentration of sulfates in the wine, which can contribute to its aroma and act as an antimicrobial agent.
- **Alcohol:** The alcohol content of the wine influences its body, flavor, and overall sensory perception.

## # import and read the data

```
readdata<-read.csv("C:/Users/Rama/Downloads/winequality-red.csv")
```

```
readdata
```

	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide	density	pH	sulphates	alcohol	quality
1	7.4	0.700	0.00	1.90	0.076	11	34	0.99780	3.51	0.56	9.4	5
2	7.8	0.880	0.00	2.60	0.098	25	67	0.99680	3.20	0.68	9.8	5
3	7.8	0.760	0.04	2.30	0.092	15	54	0.99700	3.26	0.65	9.8	5
4	11.2	0.280	0.56	1.90	0.075	17	60	0.99800	3.16	0.58	9.8	6
5	7.4	0.700	0.00	1.90	0.076	11	34	0.99780	3.51	0.56	9.4	5
6	7.4	0.660	0.00	1.80	0.075	13	40	0.99780	3.51	0.56	9.4	5
7	7.9	0.600	0.06	1.60	0.069	15	59	0.99640	3.30	0.46	9.4	5
8	7.3	0.650	0.00	1.20	0.065	15	21	0.99460	3.39	0.47	10.0	7
9	7.8	0.580	0.02	2.00	0.073	9	18	0.99680	3.36	0.57	9.5	7
10	7.5	0.500	0.36	6.10	0.071	17	102	0.99780	3.35	0.80	10.5	5
11	6.7	0.580	0.08	1.80	0.097	15	65	0.99590	3.28	0.54	9.2	5
12	7.5	0.500	0.36	6.10	0.071	17	102	0.99780	3.35	0.80	10.5	5
13	5.6	0.615	0.00	1.60	0.089	16	59	0.99430	3.58	0.52	9.9	5
14	7.8	0.610	0.29	1.60	0.114	9	29	0.99740	3.26	1.56	9.1	5
15	8.9	0.620	0.18	3.80	0.176	52	145	0.99860	3.16	0.88	9.2	5
16	8.9	0.620	0.19	3.90	0.170	51	148	0.99860	3.17	0.93	9.2	5
17	8.5	0.280	0.56	1.80	0.092	35	103	0.99690	3.30	0.75	10.5	7
18	8.1	0.560	0.28	1.70	0.368	16	56	0.99680	3.11	1.28	9.3	5
19	7.4	0.590	0.08	4.40	0.086	6	29	0.99740	3.38	0.50	9.0	4
20	7.9	0.320	0.51	1.80	0.341	17	56	0.99690	3.04	1.08	9.2	6
21	8.9	0.220	0.48	1.80	0.077	29	60	0.99680	3.39	0.53	9.4	6
22	7.6	0.390	0.31	2.30	0.082	23	71	0.99820	3.52	0.65	9.7	5
23	7.9	0.430	0.21	1.60	0.106	10	37	0.99660	3.17	0.91	9.5	5
24	8.5	0.490	0.11	2.30	0.084	9	67	0.99680	3.17	0.53	9.4	5
25	6.9	0.400	0.14	2.40	0.085	21	40	0.99680	3.43	0.63	9.7	6
26	6.3	0.390	0.16	1.40	0.080	11	23	0.99550	3.34	0.56	9.3	5
27	7.6	0.410	0.24	1.80	0.080	4	11	0.99620	3.28	0.59	9.5	5
28	7.9	0.430	0.21	1.60	0.106	10	37	0.99660	3.17	0.91	9.5	5
29	7.1	0.710	0.00	1.90	0.080	14	35	0.99720	3.47	0.55	9.4	5
30	7.8	0.645	0.00	2.00	0.082	8	16	0.99640	3.38	0.59	9.8	6

- `readdata<-read.csv("C:/Users/Rama/Downloads/winequality-red.csv")`: This line of code uses the `read.csv()` function to read the CSV file located at `"C:/Users/Rama/Downloads/winequality-red.csv"` into R. The data read from the CSV file is then stored in a variable named `readdata`. The `<-` symbol is the assignment operator in R, assigning the result of `read.csv()` to the variable `readdata`.
- `readdata`: This line of code simply prints the contents of the `readdata` variable, displaying the data read from the CSV file in the R console.

## # Data Cleaning

# To see if there is a missing value/NA in any column

```
missing_v<-is.na(readdata)
```

```
count_missing<-colSums(missing_v)
```

```
count_missing
```

```
Console Terminal Background Jobs
R 4.3.1 ~/ #
> count_missing<-colSums(missing_v)
> count_missing
      fixed.acidity      volatile.acidity      citric.acid
              0              0              0
      residual.sugar      chlorides      free.sulfur.dioxide
              0              0              0
total.sulfur.dioxide      density      pH
              0              0              0
      sulphates      alcohol      quality
              0              0              0
>
```

- `missing_v <- is.na(readdata)`: This line of code uses the `is.na()` function to create a logical matrix `missing_v` with the same dimensions as the dataset `readdata`. Each element of this matrix is `TRUE` if the corresponding element in `readdata` is missing (`NA`), and `FALSE` otherwise. This matrix essentially flags missing values in the dataset.
- `count_missing <- colSums(missing_v)`: This line of code uses the `colSums()` function to calculate the sum of `TRUE` values (missing values) in each column of the `missing_v` matrix. This gives us a count of missing values for each column in the dataset. The result is stored in the `count_missing` variable.
- `count_missing`: This line of code simply prints the contents of the `count_missing` variable, displaying the count of missing values for each column in the dataset.
- So as we can see there is no missing value in the data

**# provide a concise summary of the dataset, including basic statistics for each column.**

`summary(readdata)`

```
> summary(readdata)
fixed.acidity    volatile.acidity    citric.acid    residual.sugar
Min.   : 4.60    Min.   :0.1200    Min.   :0.000    Min.   : 0.900
1st Qu.: 7.10    1st Qu.:0.3900    1st Qu.:0.090    1st Qu.: 1.900
Median : 7.90    Median :0.5200    Median :0.260    Median : 2.200
Mean   : 8.32    Mean   :0.5278    Mean   :0.271    Mean   : 2.539
3rd Qu.: 9.20    3rd Qu.:0.6400    3rd Qu.:0.420    3rd Qu.: 2.600
Max.   :15.90    Max.   :1.5800    Max.   :1.000    Max.   :15.500

chlorides        free.sulfur.dioxide    total.sulfur.dioxide
Min.   :0.01200    Min.   : 1.00    Min.   : 6.00
1st Qu.:0.07000    1st Qu.: 7.00    1st Qu.:22.00
Median :0.07900    Median :14.00    Median :38.00
Mean   :0.08747    Mean   :15.87    Mean   :46.47
3rd Qu.:0.09000    3rd Qu.:21.00    3rd Qu.:62.00
Max.   :0.61100    Max.   :72.00    Max.   :289.00

density          pH          sulphates          alcohol
Min.   :0.9901    Min.   :2.740    Min.   :0.3300    Min.   : 8.40
1st Qu.:0.9956    1st Qu.:3.210    1st Qu.:0.5500    1st Qu.: 9.50
Median :0.9968    Median :3.310    Median :0.6200    Median :10.20
Mean   :0.9967    Mean   :3.311    Mean   :0.6581    Mean   :10.42
3rd Qu.:0.9978    3rd Qu.:3.400    3rd Qu.:0.7300    3rd Qu.:11.10
Max.   :1.0037    Max.   :4.010    Max.   :2.0000    Max.   :14.90
```

```
quality
Min.   :3.000
1st Qu.:5.000
Median :6.000
Mean   :5.636
3rd Qu.:6.000
Max.   :8.000
```

Summary function provides a summary of the main statistical measures like For numeric variables: Minimum value, 1st quartile (25th percentile), Median (50th percentile), Mean, 3rd quartile (75<sup>th</sup> percentile), Maximum value, For factor variables (categorical variables): Count of each level/category, Percentage of each level/category, For character variables: Count of each unique value, The most frequent value

**# provide information about the structure of an object, including its type and the first few elements.**

`str(readdata)`

```
R 4.3.1 ~ / > str(readdata)
'data.frame': 1599 obs. of 12 variables:
 $ fixed.acidity      : num  7.4 7.8 7.8 11.2 7.4 7.4 7.9 7.3 7.8 7.5
 ...
 $ volatile.acidity   : num  0.7 0.88 0.76 0.28 0.7 0.66 0.6 0.65 0.58
 0.5 ...
 $ citric.acid        : num  0 0 0.04 0.56 0 0 0.06 0 0.02 0.36 ...
 $ residual.sugar     : num  1.9 2.6 2.3 1.9 1.9 1.8 1.6 1.2 2 6.1 ...
 $ chlorides          : num  0.076 0.098 0.092 0.075 0.076 0.075 0.069
 0.065 0.073 0.071 ...
 $ free.sulfur.dioxide : num  11 25 15 17 11 13 15 15 9 17 ...
 $ total.sulfur.dioxide: num  34 67 54 60 34 40 59 21 18 102 ...
 $ density            : num  0.998 0.997 0.997 0.998 0.998 ...
 $ pH                : num  3.51 3.2 3.26 3.16 3.51 3.51 3.3 3.39 3.36
 3.35 ...
 $ sulphates          : num  0.56 0.68 0.65 0.58 0.56 0.56 0.46 0.47 0.
 57 0.8 ...
 $ alcohol            : num  9.4 9.8 9.8 9.8 9.4 9.4 9.4 10 9.5 10.5
 ...
 $ quality            : int  5 5 5 6 5 5 5 7 7 5 ...
>
```

The `str()` function gives you the structure of your dataset, displaying the type and first few values of each variable. This information is useful for understanding the composition of the dataset, the types of variables it contains, and the format of the data within each variable. It helps in initial data exploration and understanding the dataset before further analysis or manipulation.

**# calculate the correlation matrix between numeric variables.**

`cor(readdata)`

```
R 4.3.1 ~ / > cor(readdata)
               fixed.acidity volatile.acidity citric.acid
fixed.acidity      1.00000000      -0.256130895  0.67170343
volatile.acidity   -0.25613089      1.000000000 -0.55249568
citric.acid        0.67170343     -0.552495685  1.00000000
residual.sugar     0.11477672      0.001917882  0.14357716
chlorides          0.09370519      0.061297772  0.20382291
free.sulfur.dioxide -0.15379419     -0.010503827 -0.06097813
total.sulfur.dioxide -0.11318144      0.076470005  0.03553302
density           0.66804729      0.022026232  0.36494718
pH                -0.68297819      0.234937294 -0.54190414
sulphates          0.18300566     -0.260986685  0.31277004
alcohol           -0.06166827     -0.202288027  0.10990325
quality           0.12405165     -0.390557780  0.22637251
```

	total.sulfur.dioxide	density	pH
fixed.acidity	-0.11318144	0.66804729	-0.68297819
volatile.acidity	0.07647000	0.02202623	0.23493729
citric.acid	0.03553302	0.36494718	-0.54190414
residual.sugar	0.20302788	0.35528337	-0.08565242
chlorides	0.04740047	0.20063233	-0.26502613
free.sulfur.dioxide	0.66766645	-0.02194583	0.07037750
total.sulfur.dioxide	1.00000000	0.07126948	-0.06649456
density	0.07126948	1.00000000	-0.34169933
pH	-0.06649456	-0.34169933	1.00000000
sulphates	0.04294684	0.14850641	-0.19664760
alcohol	-0.20565394	-0.49617977	0.20563251
quality	-0.18510029	-0.17491923	-0.05773139

	sulphates	alcohol	quality
fixed.acidity	0.183005664	-0.06166827	0.12405165
volatile.acidity	-0.260986685	-0.20228803	-0.39055778
citric.acid	0.312770044	0.10990325	0.22637251
residual.sugar	0.005527121	0.04207544	0.01373164
chlorides	0.371260481	-0.22114054	-0.12890656
free.sulfur.dioxide	0.051657572	-0.06940835	-0.05065606
total.sulfur.dioxide	0.042946836	-0.20565394	-0.18510029
density	0.148506412	-0.49617977	-0.17491923
pH	-0.196647602	0.20563251	-0.05773139
sulphates	1.000000000	0.09359475	0.25139708
alcohol	0.093594750	1.000000000	0.47616632
quality	0.251397079	0.47616632	1.000000000

The `cor()` function calculates the correlation matrix between numerical variables, helping to identify relationships between variables. The resulting correlation matrix provides insights into the linear relationships between variables. The values in the correlation matrix range from -1 to 1.

#### Correlation between Citric Acid and Volatile Acidity:

- Correlation coefficient: -0.5525
- Interpretation: There is a moderate negative correlation (-0.5525) between citric acid and volatile acidity. This suggests that as the amount of citric acid increases in the wine, the volatile acidity tends to decrease, and vice versa.

#### Correlation between Fixed Acidity and Density:

- Correlation coefficient: 0.6680
- Interpretation: There is a strong positive correlation (0.6680) between fixed acidity and density. This indicates that wines with higher fixed acidity tend to have higher densities, and wines with lower fixed acidity tend to have lower densities.

#### Correlation between Alcohol and Quality:

- Correlation coefficient: 0.4762
- Interpretation: There is a moderate positive correlation (0.4762) between alcohol content and wine quality. This suggests that wines with higher alcohol content tend to have higher quality ratings, and wines with lower alcohol content tend to have lower quality ratings.

#### Correlation between Chlorides and Sulphates:

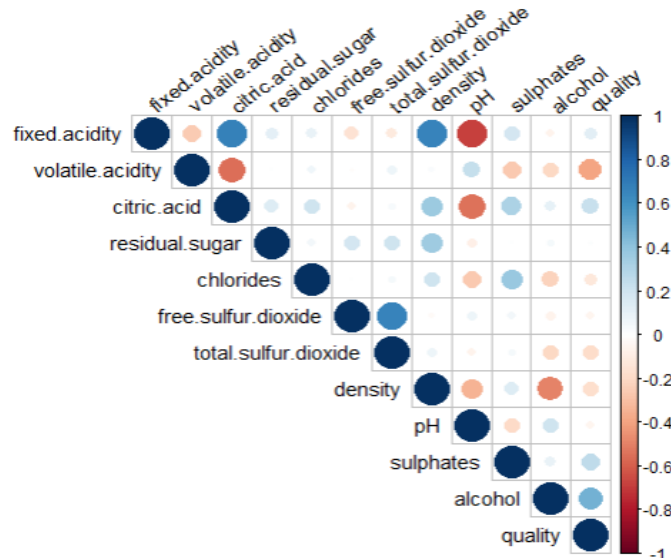
- Correlation coefficient: 0.3713
- Interpretation: There is a moderate positive correlation (0.3713) between chlorides and sulphates. This implies that as the amount of chlorides in the wine increases, the amount of sulphates also tends to increase, and vice versa.

#### Correlation between pH and Fixed Acidity:

- Correlation coefficient: -0.6830
- Interpretation: There is a strong negative correlation (-0.6830) between pH and fixed acidity. This suggests that wines with higher fixed acidity tend to have lower pH levels, and wines with lower fixed acidity tend to have higher pH levels.

## # make a corrplot of correlation matrix

```
cor_data<-cor(readdata)
library(corrplot)
corrplot(cor_data,type="upper",order="original",tl.col="black",tl.srt=40)
```



- *corrplot* is used if we want to visualize the correlation matrix using a heatmap for better interpretation
- `cor_data <- cor(readdata)`: This line calculates the correlation matrix between the numeric variables in the `readdata` dataset using the `cor()` function and assigns it to the variable `cor_data`.
- `library(corrplot)`: This line loads the `corrplot` package, which provides functions for visualizing correlation matrices.
- `corrplot(cor_data, type = "upper", order = "original", tl.col = "black", tl.srt = 40)`: This line generates a correlation plot using the `corrplot()` function. Here's what each argument does:
- `cor_data`: The correlation matrix computed earlier.
- `type = "upper"`: Specifies that only the upper triangular part of the correlation matrix will be plotted. This is often preferred to avoid redundancy, as the lower triangular part is symmetrically the same.
- `order = "original"`: Preserves the original order of variables in the dataset when plotting the correlation matrix.
- `tl.col = "black"`: Sets the color of the text labels to black.
- `tl.srt = 40`: Adjusts the rotation angle of the text labels to 40 degrees, making them easier to read when there are many variables.

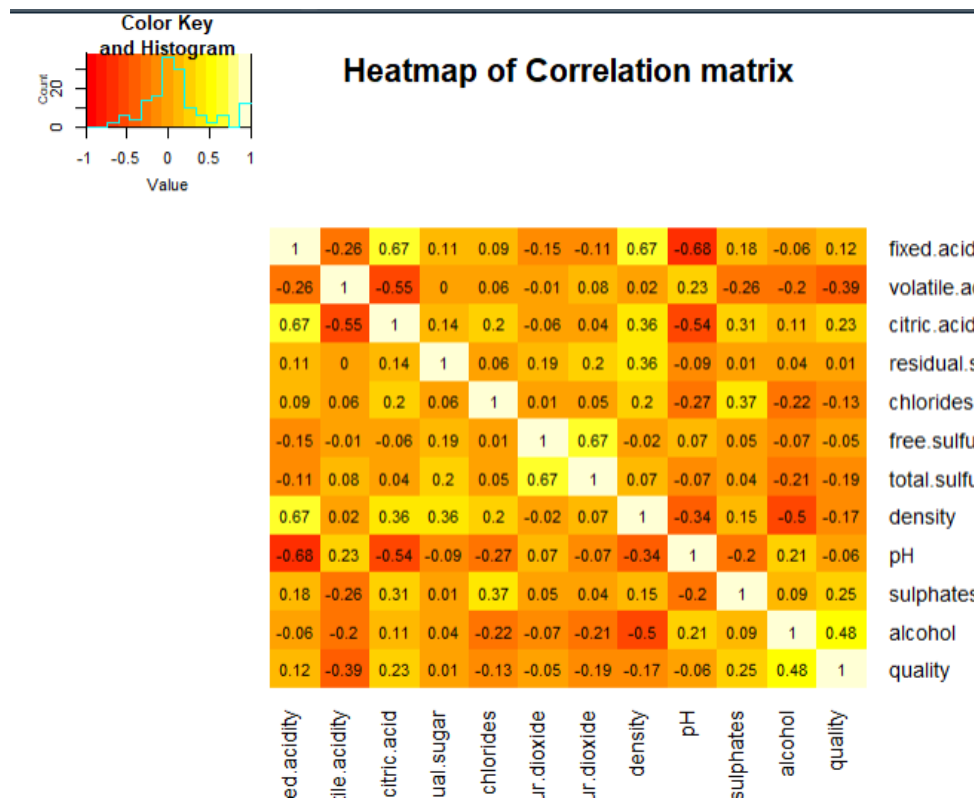
*This graph shows that there is high positive correlation between fixed acidity & citric acid or fixed acidity & density, while high negative correlation between fixed acidity & PH, citric acid & PH, citric acid & volatile acidity*

## # make a heatmap of the correlation data

```
rounded_data<-round(cor_data,2)
```



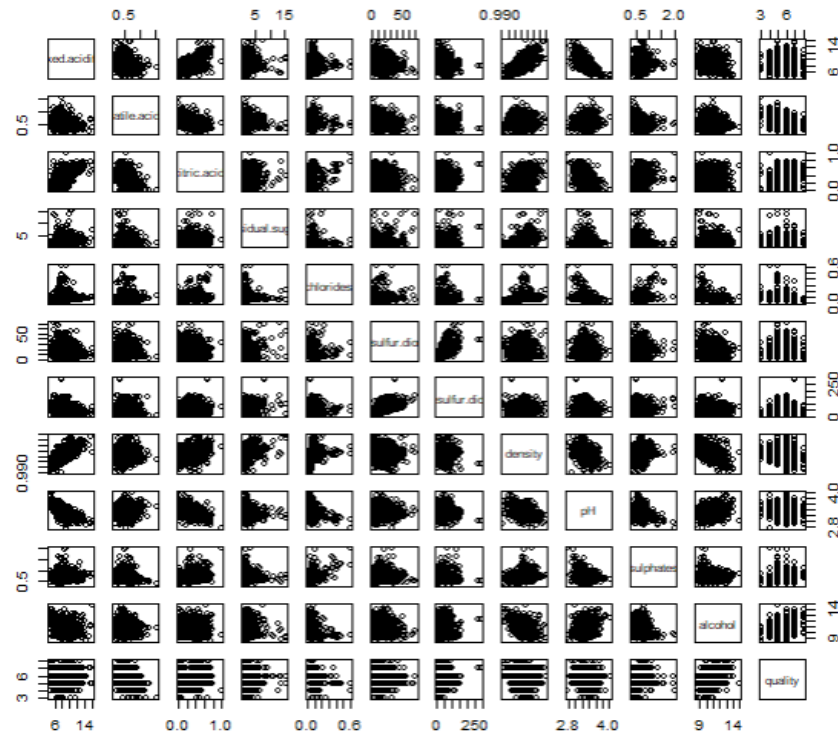
```
library(corrplot)
install.packages("gplots")
library(gplots)
heatmap.2(rounded_data,Rowv = FALSE, Colv = FALSE,dendrogram = "none",
  scale = "none",trace = "none", cellnote = rounded_data, notecol = "black", notecex = 0.9,
  main ="Heatmap of Correlation matrix")
```



- `rounded_data <- round(cor_data, 2)`: This line rounds the correlation coefficients in the `cor_data` correlation matrix to two decimal places and assigns the result to a new variable named `rounded_data`. Rounding the values can make the heatmap display cleaner and easier to interpret.
- `library(corrplot)`: This line loads the `corrplot` package, which provides functions for visualizing correlation matrices.
- `install.packages("gplots")`: This line installs the `gplots` package if it's not already installed. The `gplots` package contains the `heatmap.2` function for creating heatmaps.
- `library(gplots)`: This line loads the `gplots` package, which provides functions for graphical displays like heatmaps.
- `heatmap.2(...)`: This line generates a heatmap of the correlation matrix `rounded_data` using the `heatmap.2` function. Here's a breakdown of the arguments used:
  - `rounded_data`: The rounded correlation matrix.
  - `Rowv = FALSE, Colv = FALSE`: These arguments specify that no hierarchical clustering should be applied to rows or columns.
  - `dendrogram = "none"`: This argument suppresses the display of dendrograms.
  - `scale = "none"`: It specifies that no scaling should be applied to the data.
  - `trace = "none"`: This argument suppresses the trace lines.
  - `cellnote = rounded_data`: This argument displays the correlation coefficients as cell annotations on the heatmap.
  - `notecol = "black", notecex = 0.9`: These arguments set the color and size of the correlation coefficient annotations.
  - `main = "Heatmap of Correlation matrix"`: This argument specifies the title of the heatmap.

# Create grid of scatterplots showing the relationships between pairs of variables in a dataset.

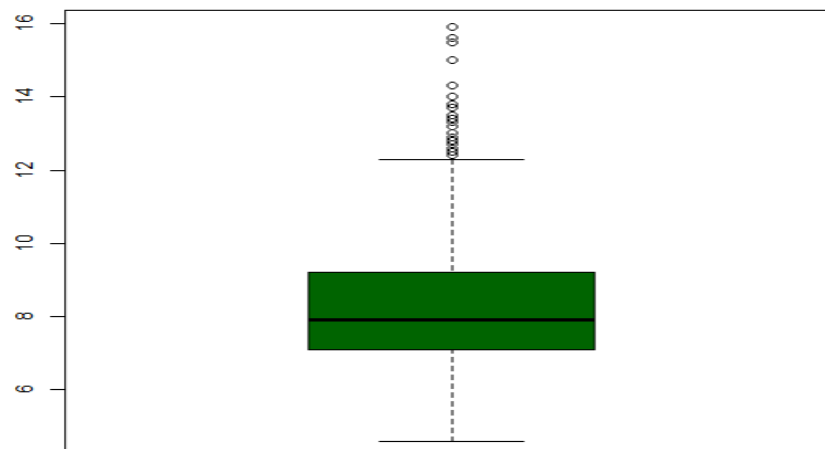
```
pairs(readdata)
```



The `pairs()` function creates a scatterplot matrix, allowing you to visualize relationships between pairs of variables in one plot. R creates a scatterplot matrix where each cell represents the relationship between two variables. Along the diagonal of the matrix, histograms for each variable are displayed. The off-diagonal cells show scatterplots of the relationships between pairs of variables. This visualization is helpful for quickly identifying potential patterns, correlations, and outliers in the data.

# Use `boxplot()` to visually identify and remove outliers

```
boxplot(readdata$fixed.acidity,col="darkgreen",border = "black")
```





- `boxplot()`: This function creates a boxplot to visualize the distribution of a numeric variable.
- `readdata$fixed.acidity`: This part of the code specifies the variable "fixed acidity" from the `readdata` dataset to be plotted in the boxplot. `$` is used to access a specific column in a data frame.
- `col = "darkgreen"`: This argument sets the fill color of the boxplot to "darkgreen".
- `border = "black"`: This argument sets the color of the boxplot borders to "black".
- This boxplot shows that there are a lot of outliers and we have to remove them.

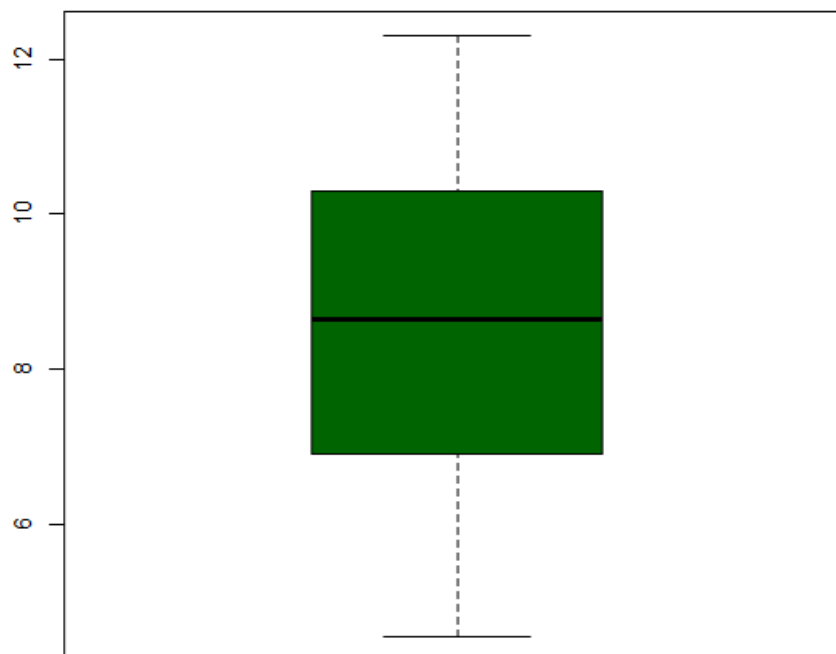
### # remove outliers using the Interquartile Range (IQR) method

```
Q1 <- quantile(readdata$fixed.acidity, 0.25)
Q3 <- quantile(readdata$fixed.acidity, 0.75)
IQR <- Q3 - Q1
```

```
# Set the threshold for outliers (typically 1.5)
threshold <- 1.5
```

```
# Identify outliers
outliers <- data < (Q1 - threshold * IQR) | data > (Q3 + threshold * IQR)
```

```
# Remove outliers
cleaned_data_fixed.acidity <- data[!outliers]
cleaned_data_fixed.acidity
# make boxplot again to verify the data is being removed
boxplot(as.numeric(cleaned_data_fixed.acidity), col="darkgreen",border = "black")
```



- `Q1, Q3, and IQR`: These variables compute the first quartile, third quartile, and interquartile range of the "fixed acidity" variable, respectively, using the `quantile()` function.
- `threshold`: This variable sets the threshold for identifying outliers. A common practice is to use 1.5 times the IQR as the threshold.
- `outliers`: This logical vector identifies outliers based on whether the "fixed acidity" values are below `Q1 - threshold * IQR` or above `Q3 + threshold * IQR`.
- `cleaned_data_fixed.acidity`: This variable stores the cleaned data, excluding the rows containing outliers.
- `Printing cleaned_data_fixed.acidity`: This line prints the cleaned dataset without outliers.
- `boxplot()`: This function creates a boxplot of the cleaned "fixed acidity" data to verify that the outliers have been removed.

- *This method has remove all the outliers now*

**# To remove majority of the outlier we can use mutate function in dplyr library to remove outlier in one go**

```
library(dplyr)
```

```
# Function to remove outliers from each column using Z-score
```

```
remove_outliers_zscore <- function(data, threshold) {  
  readdata %>%  
    mutate(across(everything(), ~ifelse(abs((. - mean(.)) / sd(.)) > threshold, NA, .)))  
}
```

```
# Set threshold for Z-score method
```

```
threshold <- 3
```

```
# Remove outliers using Z-score method
```

```
cleaned_red_wine_data <- remove_outliers_zscore(red_wine_data, threshold)  
cleaned_red_wine_data <- na.omit(cleaned_red_wine_data)
```

- *remove\_outliers\_zscore <- function(data, threshold) { ... }:* This line defines a function named `remove_outliers_zscore` that takes two arguments: `data`, which represents the dataset, and `threshold`, which sets the threshold for identifying outliers based on Z-scores.
- *readdata %>% mutate(across(everything(), ~ifelse(abs((. - mean(.)) / sd(.)) > threshold, NA, .)))*: Within the function, the dataset `readdata` is piped (`%>%`) into the `mutate()` function from the `dplyr` package. The `across()` function is used to apply the subsequent operation to all columns (`everything()`). For each column, the Z-score is calculated for each value  $(. - \text{mean}(.)) / \text{sd}(.)$ . If the absolute Z-score exceeds the specified threshold, the value is replaced with `NA`, indicating an outlier.
- *threshold <- 3:* This line sets the threshold for identifying outliers using the Z-score method. A typical threshold value is 3, but it can be adjusted based on the specific dataset and requirements.
- *cleaned\_red\_wine\_data <- remove\_outliers\_zscore(red\_wine\_data, threshold):* This line applies the `remove_outliers_zscore()` function to the `red_wine_data` dataset, removing outliers based on the specified threshold.
- *cleaned\_red\_wine\_data <- na.omit(cleaned\_red\_wine\_data):* After removing outliers, this line removes any rows containing `NA` values from the cleaned dataset using the `na.omit()` function, ensuring that the dataset does not contain missing values after outlier removal.

**# calculate the average alcohol content for each quality level in the cleaned red wine dataset.**

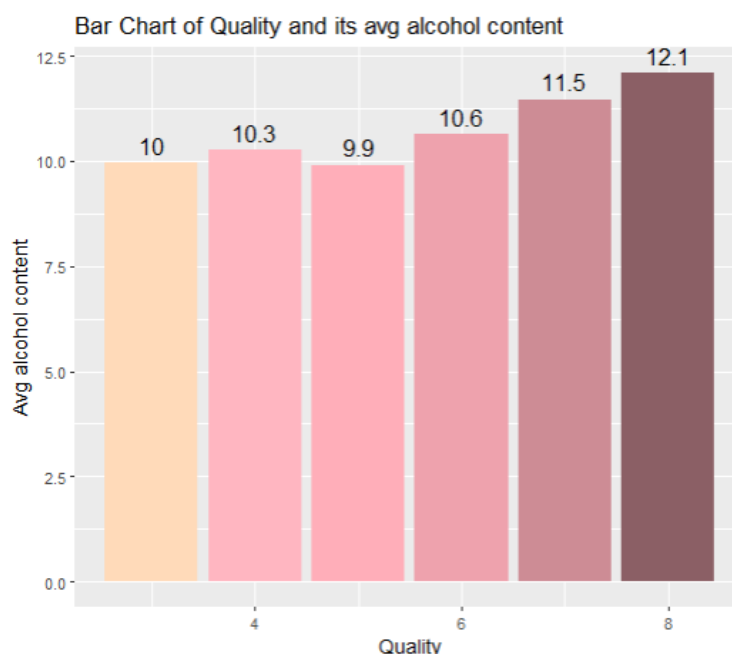
```
quality_wise_mean_alcohol <- cleaned_red_wine_data %>%  
  group_by(quality) %>%  
  summarise(avg_alcohol = mean(alcohol))
```

```
+ summarise(avg_alcohol = mean(alcohol))
# A tibble: 6 × 2
  quality avg_alcohol
  <int>     <dbl>
1     3      9.96
2     4     10.3
3     5      9.90
4     6     10.6
5     7     11.5
6     8     12.1
```

- `cleaned_red_wine_data %>%`: This pipes the cleaned red wine dataset into the subsequent operations, allowing for data manipulation using the dplyr package's syntax.
- `group_by(quality) %>%`: This groups the data by the "quality" variable. This means that subsequent operations will be applied separately to each group defined by the unique values of the "quality" variable.
- `summarise(avg_alcohol = mean(alcohol))`: Within each quality group, this calculates the mean (average) alcohol content using the `mean()` function applied to the "alcohol" variable. The result is stored in a new column named "avg\_alcohol".
- This result shows that quality 8 has the highest average alcohol while quality 5 has the lowest.

**#utilize ggplot2 to create a bar chart visualizing the average alcohol content across different quality levels of red wine.**

```
library(ggplot2)
ggplot(quality_wise_mean_alcohol,aes(x=quality,y=avg_alcohol))+
  geom_bar(stat="identity",fill=c("peachpuff","lightpink", "lightpink1","lightpink2","lightpink3",
"lightpink4"))+
  geom_text(aes(label =round(avg_alcohol,1), vjust = -0.5, size = 2)) +
  labs(title = "Bar Chart of Quality and its avg alcohol content", x = "Quality", y = "Avg alcohol
content")
```



- `ggplot(quality_wise_mean_alcohol, aes(x = quality, y = avg_alcohol))`: This initializes a ggplot object and specifies the aesthetics (aes) mapping. The "quality" variable is mapped to the x-axis, and the "avg\_alcohol" variable is mapped to the y-axis.
- `geom_bar(stat = "identity", fill = c("peachpuff", "lightpink", ...))`: This adds bars to the plot. `stat = "identity"` tells ggplot that the height of the bars should correspond to the values in the data. The fill argument assigns different fill colors to each bar based on their quality level.
- `geom_text(aes(label = round(avg_alcohol, 1), vjust = -0.5, size = 2))`: This adds text labels to the top of each bar. The label argument specifies the text to be displayed (the rounded average alcohol content), `vjust` adjusts the vertical position of the text above the bars, and `size` adjusts the size of the text.
- `labs(title = "...", x = "...", y = "...")`: This sets the titles for the plot, x-axis, and y-axis, respectively.

### # count the no. of observations in quality column

```
quality_counts <- table(cleaned_red_wine_data$quality)
quality_counts
```

```

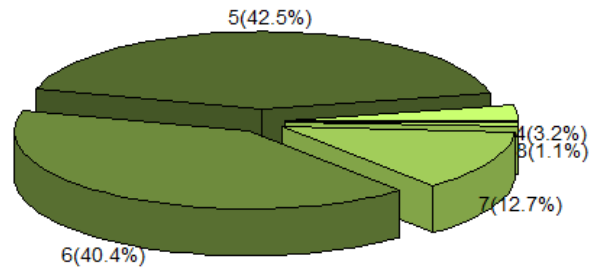
  4    5    6    7    8
53 681 638 199  18
> |
```

- `cleaned_red_wine_data$quality`: This extracts the "quality" column from the cleaned red wine dataset. This column contains the quality ratings of the red wine samples.
- `table()`: This function generates a frequency table, counting the occurrences of each unique value in the specified vector (in this case, the quality ratings).
- This table shows that 5 no. quality has the highest count

### # Create a 3D pie chart using plotrix

```
library(plotrix)
# Assuming 'red_wine_data' is your dataset, and 'quality' is the column you want to plot
# Create a summary table of wine quality counts
pie_chart <- pie3D(quality_counts, explode = 0.1, col = c("darkolivegreen1",
  "darkolivegreen", "darkolivegreen4",
  "darkolivegreen3", "darkolivegreen2"),
  , main = "Red Wine Quality Distribution",
labels = paste(names(quality_counts), "(", round(quality_counts/sum(quality_counts) * 100, 1), "%)",
  sep = "" ), labelcex = 0.9, labelcol = "black")
```

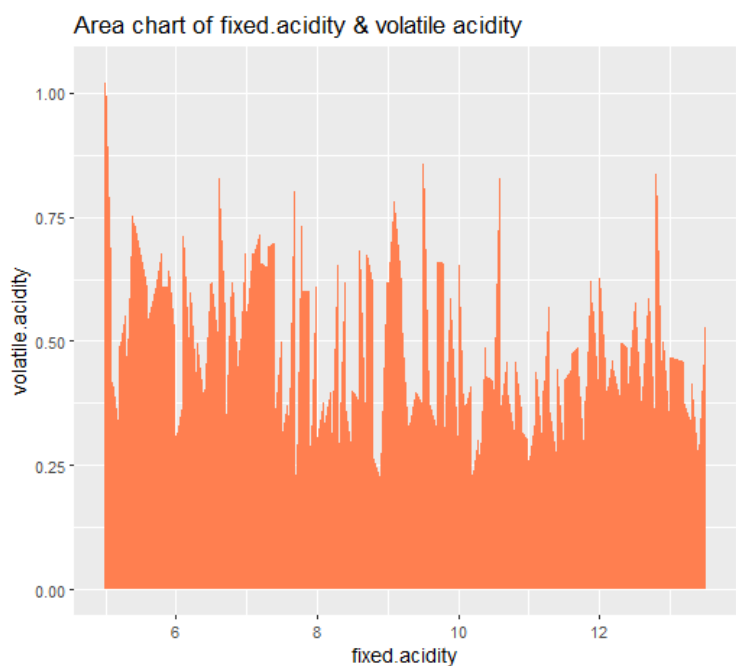
Red Wine Quality Distribution



- *pie3D()*: This function creates a 3D pie chart. It takes several arguments:
- *quality\_counts*: The table containing the counts of each quality rating.
- *explode = 0.1*: This argument controls the "exploding" effect, where slices of the pie chart are separated slightly from each other. A value of 0.1 specifies a slight separation.
- *col*: This specifies the colors for each slice of the pie chart.
- *main*: This sets the main title of the plot.
- *labels*: This argument generates labels for each slice of the pie chart. It concatenates the quality rating names with their corresponding percentages, rounded to one decimal place.
- *labelcex*: This controls the size of the labels.
- *labelcol*: This sets the color of the labels.
- This pie chart shows that 5 has the highest percentage count followed by 6,7,8,4

**#utilize ggplot2 to create an area chart visualizing the relationship between "fixed acidity" and "volatile acidity"**

```
ggplot(cleaned_red_wine_data,aes(x=fixed.acidity,y=volatile.acidity))+
  geom_area(fill="coral")+
  labs(title = "Area chart of fixed.acidity & volatile acidity")
```



- `library(ggplot2)`: This line loads the ggplot2 package, which is used for data visualization.
- `ggplot(cleaned_red_wine_data, aes(x = fixed.acidity, y = volatile.acidity))`: This initializes a ggplot object and specifies the aesthetics (aes) mapping. The "fixed acidity" variable is mapped to the x-axis, and the "volatile acidity" variable is mapped to the y-axis.
- `geom_area(fill = "coral")`: This adds an area layer to the plot. The fill argument sets the fill color of the area to "coral". The area chart represents the relationship between "fixed acidity" and "volatile acidity" by filling the area under the curve created by connecting the data points.
- `labs(title = "Area chart of fixed.acidity & volatile acidity")`: This sets the title of the plot.

### # group data by pH level and summarize by avg total sulphur

```
ph_sulphur<-cleaned_red_wine_data%>%
  group_by(pH)%>%
  summarise(avg_totalsulfur=mean(total.sulfur.dioxide))
ph_sulphur
```

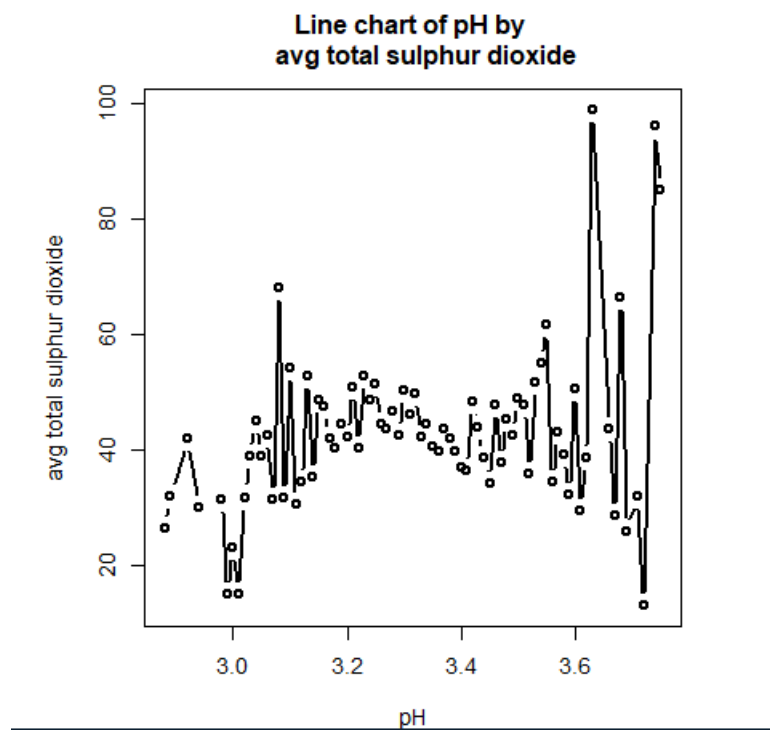
```
> ph_sulphur
# A tibble: 85 × 2
   pH avg_totalsulfur
  <dbl>          <dbl>
1  2.86             15
2  2.87          103
3  2.88          26.5
4  2.89          70.5
5  2.9             64
6  2.92          27.8
7  2.93          135
8  2.94          54.2
9  2.95           43
10 2.98          39.4
# i 75 more rows
# i Use `print(n = ...)` to see more rows
>
```

- `ph_sulphur <- cleaned_red_wine_data %>%`: This assigns a new variable `ph_sulphur` and pipes (`%>%`) the `cleaned_red_wine_data` dataframe into the next operation.
- `group_by(pH) %>%`: This groups the data by the pH variable, meaning that calculations will be performed on subsets of the data where the values of pH are the same.
- `summarise(avg_totalsulfur = mean(total.sulfur.dioxide))`: This calculates the mean of the `total.sulfur.dioxide` variable within each group of pH. It creates a new column called `avg_totalsulfur` to store these mean values.
- `ph_sulphur`: This line simply prints the resulting dataframe `ph_sulphur`, which contains the average total sulfur dioxide levels grouped by pH.
- PH level 2.93 has the highest avg total sulphur of 135 whereas level 2.86 has the lowest of 15.

### # plot the line chart of avg total sulphur

```
plot(ph_sulphur, type = "b", lwd=2, main="Line chart of pH by
  avg total sulphur dioxide ", ylab="avg total sulphur dioxide")
```





- `ph_sulphur <- cleaned_red_wine_data %>%`: This assigns a new variable `ph_sulphur` and pipes (`%>%`) the `cleaned_red_wine_data` dataframe into the next operation.
- `group_by(pH) %>%`: This groups the data by the `pH` variable, meaning that calculations will be performed on subsets of the data where the values of `pH` are the same.
- `summarise(avg_totalsulfur = mean(total.sulfur.dioxide))`: This calculates the mean of the `total.sulfur.dioxide` variable within each group of `pH`. It creates a new column called `avg_totalsulfur` to store these mean values.
- `ph_sulphur`: This line simply prints the resulting dataframe `ph_sulphur`, which contains the average total sulfur dioxide levels grouped by `pH`

**# group the data by quality and find the maximum free sulphur dioxide for each quality**

```
sulphur_max_quality<-cleaned_red_wine_data%>%
group_by(quality)%>%
summarise(max_free_sulphur=max(free.sulfur.dioxide))
```

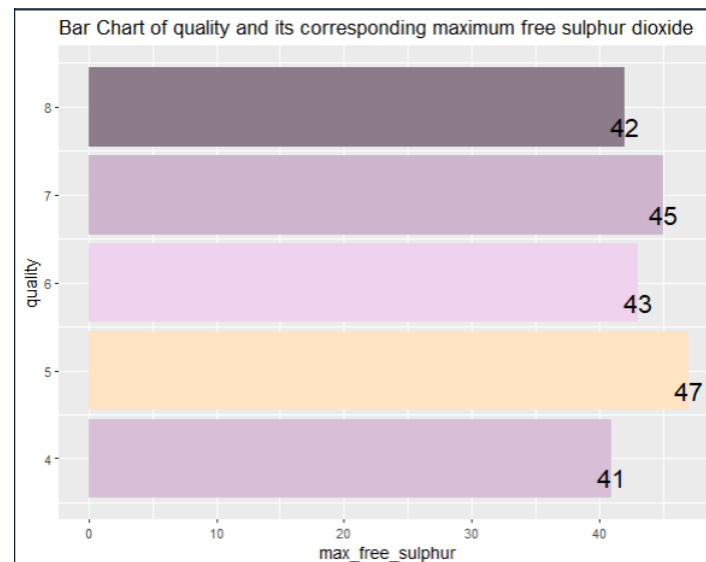
```
> sulphur_max_quality
# A tibble: 5 × 2
  quality max_free_sulphur
  <int>         <dbl>
1     4             41
2     5             47
3     6             43
4     7             45
5     8             42
>
```

- `sulphur_max_quality <- cleaned_red_wine_data %>%`: This assigns a new variable `sulphur_max_quality` and pipes (`%>%`) the `cleaned_red_wine_data` dataframe into the next operation.
- `group_by(quality) %>%`: This groups the data by the `quality` variable, meaning that calculations will be performed on subsets of the data where the values of `quality` are the same.

- `summarise(max_free_sulphur = max(free.sulfur.dioxide))`: This calculates the maximum value of the `free.sulfur.dioxide` variable within each group of quality. It creates a new column called `max_free_sulphur` to store these maximum values.
- Here the maximum sulphur dioxide is for quality 5 and minimum for quality 4

**# utilize `ggplot2` in R to create a bar chart illustrating the relationship between wine quality and its corresponding maximum free sulfur dioxide.**

```
ggplot(sulphur_max_quality,aes(x=quality,y =max_free_sulphur))+
  geom_bar(fill = c("thistle","bisque","thistle2","thistle3","thistle4"),stat = "identity")+
  ggtitle("Bar Chart of quality and its corresponding maximum free sulphur dioxide")+
  geom_text(aes(label = max_free_sulphur), size = 6,vjust = 1.5, position = position_dodge(.9))+
  coord_flip()
```



- `ggplot`: Initializes the plot with the data `sulphur_max_quality` and aesthetics mapping `quality` to the x-axis and `max_free_sulphur` to the y-axis.
- `geom_bar`: Plots the bars with colors specified by the `fill` argument. `stat = "identity"` is used to indicate that the height of the bars should be taken from the data.
- `ggtitle`: Adds a title to the plot.
- `geom_text`: Adds text labels to each bar, with the label values being taken from the `max_free_sulphur` column. `position_dodge(.9)` is used to position the labels dodging each other slightly for better readability.
- `coord_flip`: Flips the coordinates to create a horizontal bar chart instead of the default vertical orientation.

**# group data by residual sugar and then arrange chloride in descending order**

```
cumsum_chloride<-cleaned_red_wine_data%>%
  group_by(residual.sugar)%>%
  arrange(desc(chlorides))
cumsum_chloride1<-cumsum_chloride[,c(4,5)]
cumsum_chloride2<-head(cumsum_chloride1)
cumsum_chloride3<-cumsum_chloride2[c(1:5),]
cumsum_chloride3
```

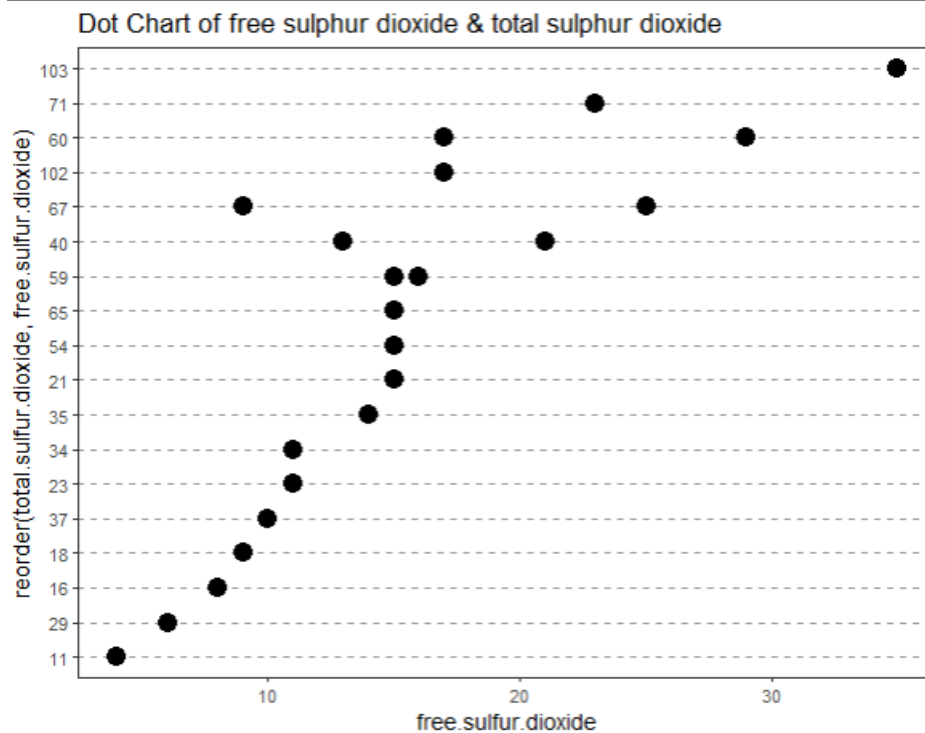
	residual.sugar	chlorides
	<dbl>	<dbl>
1	2.5	0.226
2	2.5	0.226
3	1.8	0.222
4	2.1	0.216
5	2.4	0.214

>

- `cumsum_chloride <- cleaned_red_wine_data %>% group_by(residual.sugar) %>% arrange(desc(chlorides))`: This code first groups the data by the `residual.sugar` column, then arranges the rows within each group in descending order based on the `chlorides` column.
- `cumsum_chloride1 <- cumsum_chloride[,c(4,5)]`: This line selects columns 4 and 5 (presumably the `residual.sugar` and `chlorides` columns) from the `cumsum_chloride` dataframe and assigns the result to `cumsum_chloride1`.
- `cumsum_chloride2 <- head(cumsum_chloride1)`: This line selects the first few rows (by default, 6 rows) of the `cumsum_chloride1` dataframe and assigns the result to `cumsum_chloride2`.
- `cumsum_chloride3 <- cumsum_chloride2[c(1:5),]`: This line selects rows 1 through 5 from the `cumsum_chloride2` dataframe and assigns the result to `cumsum_chloride3`.
- `cumsum_chloride3`: This line simply prints the resulting dataframe `cumsum_chloride3`, which contains the selected rows and columns from the original data.
- Here 2.5 has the highest value while 2.4 has the lowest values

**# utilize `ggplot2` to create a dot chart (scatter plot) comparing free sulfur dioxide and total sulfur dioxide for the top 25 rows of the `cleaned_red_wine_data` dataset.**

```
data_top25 <- cleaned_red_wine_data[1:25, ]
# Take the top 25 from the tophitters data set
ggplot(data_top25, aes(x = free.sulfur.dioxide, y=reorder(total.sulfur.dioxide,free.sulfur.dioxide))) +
  geom_point(size = 4) + # Use a larger dot
  ggtitle("Dot Chart of free sulphur dioxide & total sulphur dioxide")+
  theme_bw() +
  theme(
    panel.grid.major.x = element_blank(),
    panel.grid.minor.x = element_blank(),
    panel.grid.major.y = element_line(colour = "grey60", linetype = "dashed")
  )
```



- `data_top25 <- cleaned_red_wine_data[1:25, ]`: This line creates a new dataframe `data_top25` containing the first 25 rows of the `cleaned_red_wine_data`.
- `ggplot(data_top25, aes(x = free.sulfur.dioxide, y = reorder(total.sulfur.dioxide, free.sulfur.dioxide)))` `+`: This initializes the plot with the data from `data_top25`, mapping `free.sulfur.dioxide` to the x-axis and `total.sulfur.dioxide` to the y-axis. The `reorder()` function is used to reorder the `total.sulfur.dioxide` variable based on the `free.sulfur.dioxide` variable.
- `geom_point(size = 4)`: This adds points to the plot, representing each data point. `size = 4` is used to specify the size of the points.
- `ggtitle("Dot Chart of free sulphur dioxide & total sulphur dioxide")`: This adds a title to the plot.
- `theme_bw()`: This sets the plot theme to a simple black and white theme.
- `theme(...)`: This further customizes the theme by removing major grid lines on the x-axis (`panel.grid.major.x = element_blank()`), removing minor grid lines on the x-axis (`panel.grid.minor.x = element_blank()`), and adding dashed grey grid lines on the y-axis (`panel.grid.major.y = element_line(colour = "grey60", linetype = "dashed")`).

**# use data.table package to convert a dataframe (cleaned\_red\_wine\_data) to a data.table (red\_wine\_data\_dt) and then perform operations on it.**

```
library(data.table)
# Convert your big mart data to a data.table (assuming it's in a data frame called 'big_mart_data')
red_wine_data_dt <- as.data.table(cleaned_red_wine_data)
# Perform operations
red_wine_data_dt[, .(Avg_alcohol = mean(alcohol)), by = .(fixed.acidity,
volatile.acidity, residual.sugar)]
```

```

> red_wine_data_dt[, (Avg_alcohol = mean(alcohol)), by = .(fixed.acidity,
y, volatile.acidity, residual.sugar)]
      fixed.acidity volatile.acidity residual.sugar Avg_alcohol
      <num>          <num>          <num>          <num>
1:           7.4          0.700           1.9           9.4
2:           7.8          0.880           2.6           9.8
3:           7.8          0.760           2.3           9.8
4:          11.2          0.280           1.9           9.8
5:           7.4          0.660           1.8           9.4
---
1202:          6.8          0.620           1.9           9.5
1203:          6.2          0.600           2.0          10.5
1204:          5.9          0.550           2.2          11.2
1205:          5.9          0.645           2.0          10.2
1206:          6.0          0.310           3.6          11.0
>

```

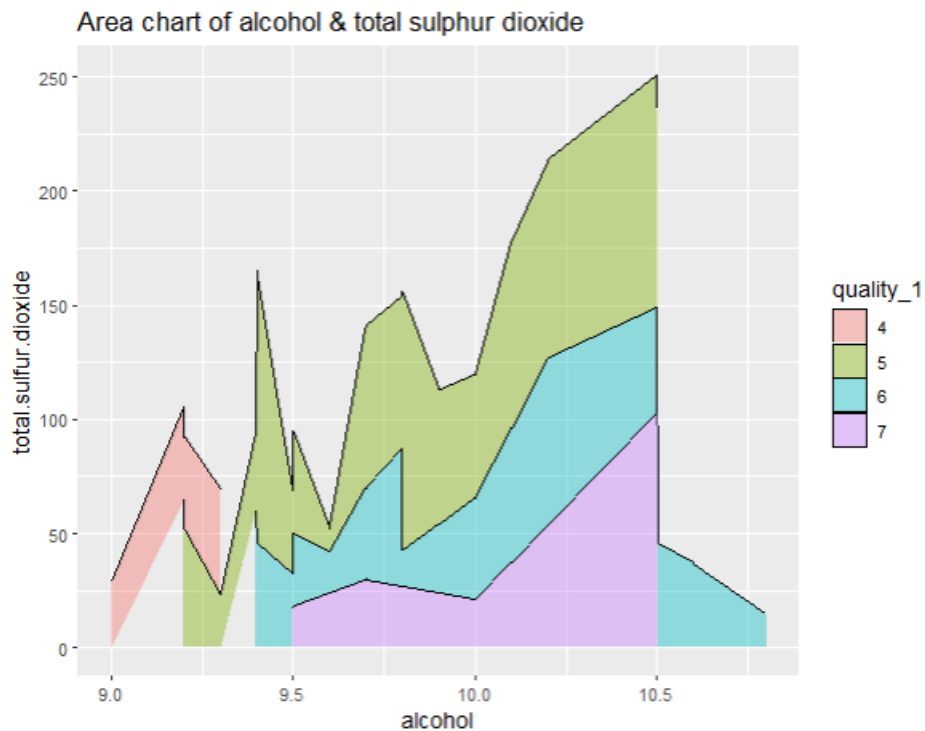
- `library(data.table)`: This loads the `data.table` package, which provides enhanced functionality for working with data in R.
- `red_wine_data_dt <- as.data.table(cleaned_red_wine_data)`: This line converts the dataframe `cleaned_red_wine_data` to a `data.table` and assigns it to the variable `red_wine_data_dt`.
- `red_wine_data_dt[, .(Avg_alcohol = mean(alcohol)), by = .(fixed.acidity, volatile.acidity, residual.sugar)]`: This performs an operation on the `red_wine_data_dt` `data.table`. It calculates the mean of the alcohol column, grouping the data by the columns `fixed.acidity`, `volatile.acidity`, and `residual.sugar`. The result is a `data.table` with columns `fixed.acidity`, `volatile.acidity`, `residual.sugar`, and `Avg_alcohol`, representing the grouped average alcohol content for each combination of the specified variables.

**# use the `ggplot2` package to create an area chart representing the relationship between alcohol content (`alcohol`) and total sulfur dioxide (`total.sulfur.dioxide`) in the top 50 values of the cleaned red wine dataset.**

```

quality_1<-as.character(top_50_values$quality)
top_50_values<-cleaned_red_wine_data[1:50,]
ggplot(top_50_values, aes(x = alcohol, y =total.sulfur.dioxide, fill =quality_1)) +
  geom_area(colour = "black", size = .2, alpha = .4) +
  ggtitle("Area chart of alcohol & total sulphur dioxide")+
  scale_fill_brewer(palette = "Reds")

```

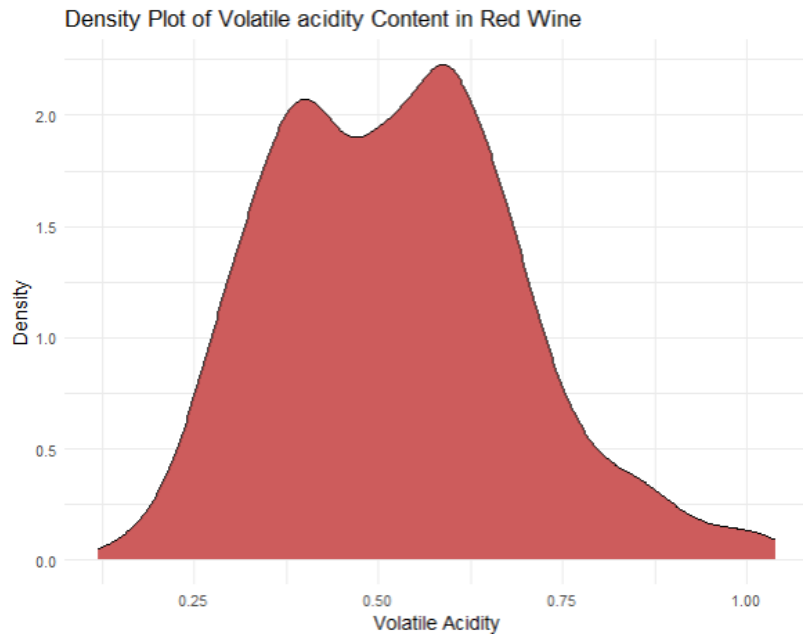


- `quality_1 <- as.character(top_50_values$quality)`: This line extracts the quality column from the `top_50_values` dataframe and converts it to a character vector. This variable is used to fill the areas in the plot with different colors based on quality.
- `top_50_values <- cleaned_red_wine_data[1:50, ]`: This line subsets the `cleaned_red_wine_data` dataframe to contain only the first 50 rows and assigns it to the `top_50_values` variable.
- `ggplot(top_50_values, aes(x = alcohol, y = total.sulfur.dioxide, fill = quality_1))` `+`: This initializes a ggplot object using the `top_50_values` dataframe as the data source and sets the aesthetics (`aes`). It maps `alcohol` to the x-axis, `total.sulfur.dioxide` to the y-axis, and `quality_1` to the fill color.
- `geom_area(colour = "black", size = .2, alpha = .4)` `+`: This adds the area layer to the plot. The `colour` parameter sets the color of the border lines around the areas, `size` sets the width of the border lines, and `alpha` sets the transparency of the areas.
- `ggtitle("Area chart of alcohol & total sulphur dioxide ")` `+`: This sets the title of the plot.
- `scale_fill_brewer(palette = "Reds")`: This function sets the fill color palette to "Reds". It ensures that each level of the `quality_1` variable will be represented by a different shade of red.

**# use `ggplot2` to create a density plot of the volatile acidity content in red wine.**

```
ggplot(cleaned_red_wine_data, aes(x = volatile.acidity)) +
  geom_density(fill = "indianred", color = "black") + # Density plot with indianred fill and black border
  labs(title = "Density Plot of Volatile acidity Content in Red Wine", x = "Volatile Acidity", y =
"Density") + # Adding title and axis labels
  theme_minimal() # Applying a minimal theme
```

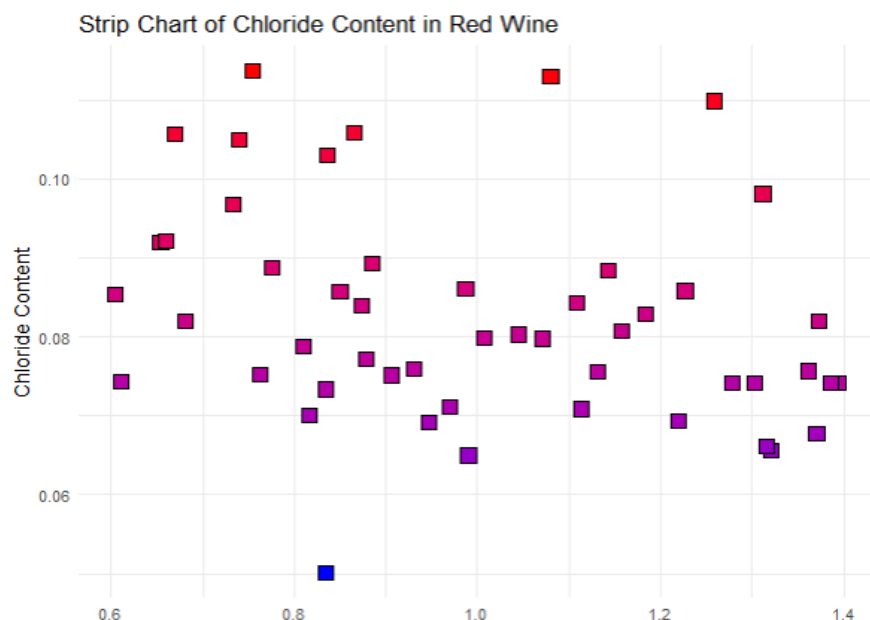




- `ggplot(cleaned_red_wine_data, aes(x = volatile.acidity))`: Initializes a `ggplot` object with `cleaned_red_wine_data` as the data source and maps the `volatile.acidity` variable to the x-axis.
- `geom_density(fill = "indianred", color = "black")`: Adds a density plot layer to the plot. The `fill` parameter sets the fill color of the density plot to "indianred", and the `color` parameter sets the border color to "black".
- `labs(title = "Density Plot of Volatile acidity Content in Red Wine", x = "Volatile Acidity", y = "Density")`: Adds a title and labels to the x-axis and y-axis respectively.
- `theme_minimal()`: Applies a minimal theme to the plot, which removes the background grid lines and minimizes unnecessary decorations.

**# create a strip chart (or jitter plot) of chloride content in red wine for the top 50 values.**

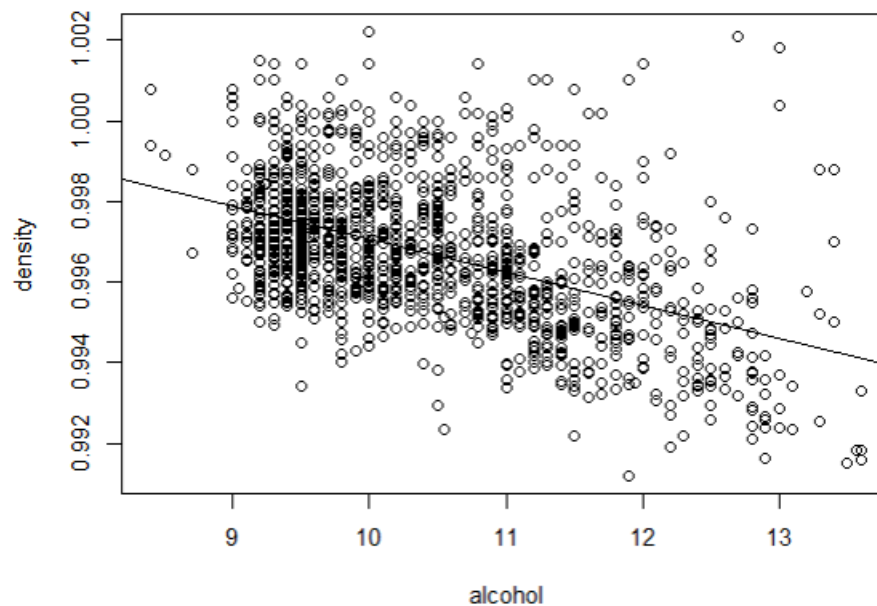
```
ggplot(top_50_values, aes(x = 1, y = chlorides, fill = chlorides)) +
  geom_jitter(shape = 22, size = 4) +
  scale_fill_gradient(low = "blue", high = "red") +
  labs(title = "Strip Chart of Chloride Content in Red Wine",
       x = "", y = "Chloride Content") +
  theme_minimal() +
  theme(legend.position = "none")
```



- `ggplot(top_50_values, aes(x = 1, y = chlorides, fill = chlorides))`: Initializes a `ggplot` object with `top_50_values` as the data source and maps the y-axis to the `chlorides` variable. The x-axis is set to 1 to create a single group of points.
- `geom_jitter(shape = 22, size = 4)`: Adds jittered points to the plot. `shape = 22` specifies solid circular points, and `size = 4` sets the size of the points.
- `scale_fill_gradient(low = "blue", high = "red")`: Applies a gradient fill from blue to red based on the values of the `chlorides` variable.
- `labs(title = "Strip Chart of Chloride Content in Red Wine", x = "", y = "Chloride Content")`: Adds a title and labels to the y-axis.
- `theme_minimal()`: Applies a minimal theme to the plot.
- `theme(legend.position = "none")`: Removes the legend from the plot.

**# perform linear regression and plots the resulting regression line on a scatter plot of density versus alcohol content.**

```
reg<-lm(density~alcohol)
abline(reg)
```



- `reg <- lm(density ~ alcohol)`: This line fits a linear regression model (`lm`) where `density` is the response variable and `alcohol` is the predictor variable.
- `abline(reg)`: This line adds a straight line to the plot representing the regression line calculated by the linear regression model `reg`.
- The regression line itself is downward sloping reflecting negative relationship between density and alcohol.

**# split the dataset into training and testing sets, then fitting a simple linear regression model to the training set.**

```
library(caTools)
# Split the data into training and testing sets
set.seed(123)
train_index <- sample.split(cleaned_red_wine_data, SplitRatio = 0.8)
train_data <- subset(cleaned_red_wine_data, train_index==TRUE)
test_data <- subset(cleaned_red_wine_data, train_index==FALSE)
# fitting simple regression to the training set
```

```
regressor=lm(formula = fixed.acidity~volatile.acidity,data=train_data)
summary(regressor)
```

```
Residuals:
    Min       1Q   Median       3Q      Max
-3.2875 -1.0432 -0.3495  0.8707  5.3884

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)      9.7507     0.1511   64.54  <2e-16 ***
volatile.acidity -2.7846     0.2725  -10.22  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.546 on 1086 degrees of freedom
Multiple R-squared:  0.08769,    Adjusted R-squared:  0.08685
F-statistic: 104.4 on 1 and 1086 DF,  p-value: < 2.2e-16

> |
```

- *library(caTools):* This line loads the *caTools* package, which contains functions for various data splitting and manipulation tasks.
- *set.seed(123):* This sets the seed for reproducibility of random processes. In this case, it ensures that the data splitting process will be the same every time you run the code, which is important for reproducibility.
- *train\_index <- sample.split(cleaned\_red\_wine\_data, SplitRatio = 0.8):* This line splits the *cleaned\_red\_wine\_data* dataset into training and testing sets using the *sample.split* function from the *caTools* package. It randomly assigns rows to either the training or testing set based on the *SplitRatio*, which is set to 0.8, meaning that 80% of the data will be used for training and 20% for testing. The resulting *train\_index* variable is a logical vector indicating which rows belong to the training set (*TRUE*) and which belong to the testing set (*FALSE*).
- *train\_data <- subset(cleaned\_red\_wine\_data, train\_index == TRUE):* This line creates the training dataset by subsetting *cleaned\_red\_wine\_data* using the *train\_index* logical vector. It selects only the rows for which *train\_index* is *TRUE*.
- *test\_data <- subset(cleaned\_red\_wine\_data, train\_index == FALSE):* This line creates the testing dataset similarly to the previous step, but it selects rows for which *train\_index* is *FALSE*.
- *regressor = lm(formula = fixed.acidity ~ volatile.acidity, data = train\_data):* This line fits a simple linear regression model to the training data. It predicts *fixed.acidity* based on *volatile.acidity*. The result is stored in the *regressor* object.
- *summary(regressor):* This line provides a summary of the linear regression model, including coefficients, standard errors, t-values, and p-values.

#### Residuals:

- *Min, 1Q, Median, 3Q, Max:* These are descriptive statistics of the residuals, which are the differences between the observed values and the values predicted by the regression model.
- *Min:* The smallest residual is -3.2875.
- *1Q (First Quartile):* 25% of the residuals fall below this value, which is -1.0432.
- *Median:* 50% of the residuals fall below this value, which is -0.3495.
- *3Q (Third Quartile):* 75% of the residuals fall below this value, which is 0.8707.
- *Max:* The largest residual is 5.3884.

#### Coefficients:

- *Estimate:*
- The estimated intercept (when all predictor variables are zero) is 9.7507.
- The estimated coefficient for *volatile.acidity* is -2.7846. This means that for every one-unit increase in *volatile.acidity*, the predicted value of the response variable (*fixed acidity*) decreases by approximately 2.7846 units.
- *Std. Error:* These are the standard errors associated with the coefficients. They indicate the precision of the coefficient estimates.

- *t value: These are the t-values associated with each coefficient. They measure the number of standard errors the coefficient is away from zero.*
- *The intercept has a t-value of 64.54.*
- *The volatile.acidity coefficient has a t-value of -10.22.*
- *Pr(>|t|): These are the p-values associated with each coefficient. They indicate the probability of observing a t-value as extreme as, or more extreme than, the observed t-value under the null hypothesis that the coefficient is equal to zero. Smaller p-values indicate stronger evidence against the null hypothesis.*

#### **Residual standard error:**

- *This is the standard deviation of the residuals, which is approximately 1.546. It represents the average distance of the observed values from the regression line.*
- *Multiple R-squared and Adjusted R-squared:*
- *Multiple R-squared: This is a measure of how well the regression model explains the variability of the response variable. In this case, it is approximately 0.08769, indicating that about 8.769% of the variability in the response variable is explained by the model.*
- *Adjusted R-squared: This is a modified version of the R-squared that adjusts for the number of predictor variables in the model. It is approximately 0.08685.*

#### **F-statistic:**

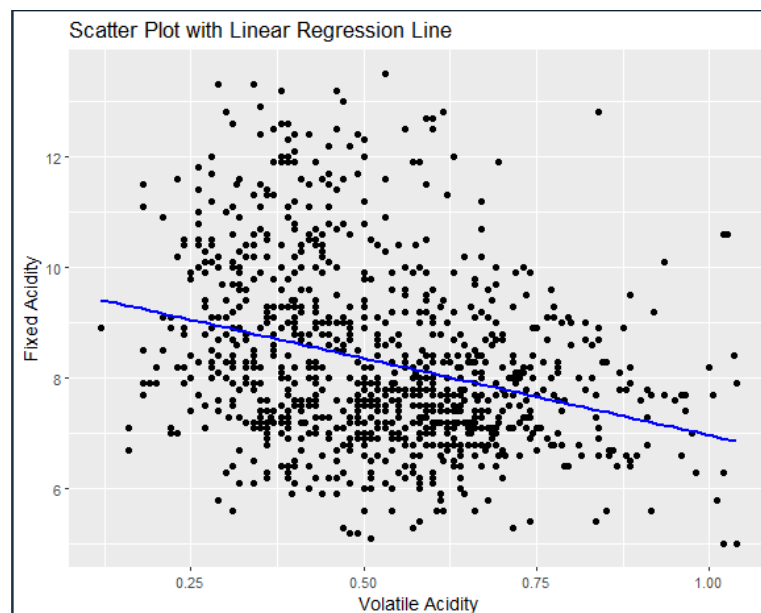
- *F-statistic: This is a test statistic for the overall significance of the regression model. It tests whether at least one of the predictor variables has a nonzero coefficient. In this case, the F-statistic is 104.4, with 1 and 1086 degrees of freedom.*
- *p-value: The p-value associated with the F-statistic is very small (less than 2.2e-16), indicating strong evidence against the null hypothesis that all coefficients are equal to zero.*

#### **Conclusion:**

- *In summary, the linear regression model suggests that volatile acidity has a significant effect on fixed acidity. The model explains a small portion of the variability in fixed acidity, as indicated by the low R-squared value. However, the overall model is statistically significant, as indicated by the small p-value associated with the F-statistic.*

### **# Create scatter plot with regression line**

```
scatter_plot <- ggplot(train_data, aes(x = volatile.acidity, y = fixed.acidity)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE, color = "blue") + # Add regression line
  labs(x = "Volatile Acidity", y = "Fixed Acidity", title = "Scatter Plot with Linear Regression Line")
# Display the plot
print(scatter_plot)
```



- `scatter_plot <- ggplot(train_data, aes(x = volatile.acidity, y = fixed.acidity))` *+: Initializes a ggplot object with the train\_data dataframe as the data source. It maps volatile.acidity to the x-axis and fixed.acidity to the y-axis.*
- `geom_point()` *+: Adds a layer of points to the plot using geom\_point(). Each point represents an observation in the dataset.*
- `geom_smooth(method = "lm", se = FALSE, color = "blue")` *+: Adds a linear regression line to the plot using geom\_smooth(). The method = "lm" argument specifies that a linear regression model should be used. se = FALSE suppresses the display of the confidence interval around the regression line. The color = "blue" argument sets the color of the regression line to blue.*
- `labs(x = "Volatile Acidity", y = "Fixed Acidity", title = "Scatter Plot with Linear Regression Line")` *: Adds labels to the x-axis (Volatile Acidity), y-axis (Fixed Acidity), and a title (Scatter Plot with Linear Regression Line) to the plot.*
- `print(scatter_plot)` *: Displays the scatter plot with the linear regression line.*
- *This graphs shows negative relationship between fixed acidity and volatile acidity as reflected by the coefficient of volatile acidity also which is negative. As one variable increases, the other tends to decrease.*

### # predict the test set result

```
y_pred=predict(regressor,newdata = test_data)
```

```
y_pred
```

```
> y_pred=predict(regressor,newdata = test_data)
> y_pred
      5      8     11     22     25     28     35
7.801452 7.940683 8.135606 8.664683 8.636837 8.553298 8.859606
      38     42     51     54     57     64     67
8.692529 8.052067 7.912836 8.692529 8.581144 7.703990 8.302683
      70     76     79     83     94     98    101
7.787529 8.608990 7.662221 8.358375 8.386221 8.358375 8.052067
      108    112    115    122    125    130    136
7.996375 8.024221 8.191298 8.219144 8.358375 8.692529 7.676144
      139    142    151    155    158    166    169
8.191298 7.759683 8.831760 8.553298 8.553298 7.996375 7.996375
      173    179    183    186    192    195    198
8.581144 7.509067 7.717913 8.887452 8.720375 8.219144 8.915298
      205    208    211    217    220    223    230
8.553298 8.163452 8.274837 8.010298 8.274837 8.052067 8.302683
      233    236    243    248    251    257    261
8.720375 7.996375 8.135606 8.079913 8.859606 8.803914 8.831760
```

- `y_pred = predict(regressor, newdata = test_data)` *: This line generates predictions (y\_pred) for the response variable (dependent variable) based on the linear regression model (regressor) using new data (test\_data) as input.*
- `y_pred` *: This line outputs the predicted values of the response variable (fixed.acidity) based on the model and the new data.*
- *The output y\_pred will contain a vector of predicted values for the response variable (fixed.acidity) based on the input data (test\_data) and the coefficients learned by the linear regression model (regressor). These predicted values represent the model's estimation of the response variable for the given predictor variables in the test\_data.*

# create a data frame named `predictions` containing the actual values of the response variable (`fixed.acidity`) from the test data (`test_data`) and the predicted values of the response variable (`y_pred`).

```
predictions <- data.frame(Actual = test_data$fixed.acidity, Predicted = y_pred)
head(predictions)
```

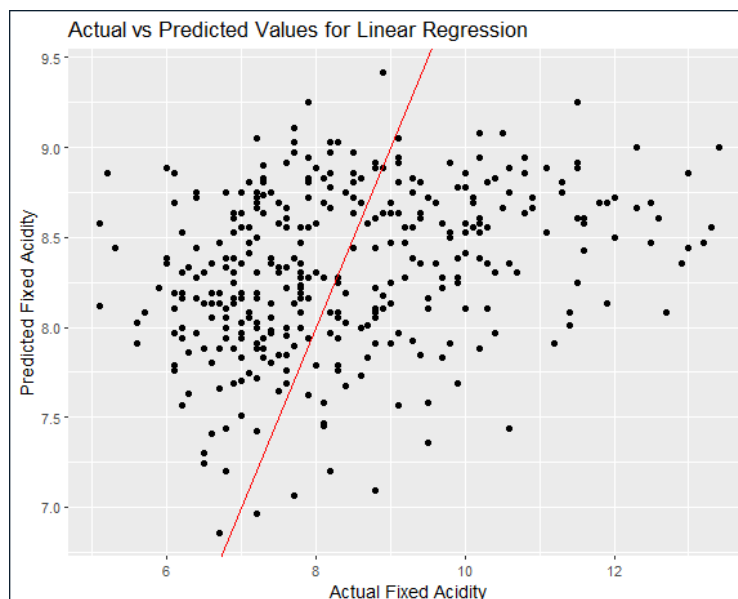
```
> head(predictions)
  Actual Predicted
5    7.4   7.801452
8    7.3   7.940683
11   6.7   8.135606
22   7.6   8.664683
25   6.9   8.636837
28   7.9   8.553298
>
```

- `predictions <- data.frame(Actual = test_data$fixed.acidity, Predicted = y_pred)`: This line creates a data frame named `predictions`. It has two columns:
- `Actual`: This column contains the actual values of the response variable (`fixed.acidity`) from the test data (`test_data$fixed.acidity`).
- `Predicted`: This column contains the predicted values of the response variable generated by the linear regression model (`y_pred`).
- `head(predictions)`: This line displays the first few rows of the `predictions` data frame.
- The `predictions` data frame provides a convenient way to compare the actual values of the response variable with the corresponding predicted values generated by the linear regression model. By examining these values, you can evaluate the performance of the model in predicting the response variable based on the predictor variables in the test data.

**# create a scatter plot comparing the actual values of the response variable (`Actual`) with the predicted values (`Predicted`) generated by the linear regression model.**

```
scatter_plot <- ggplot(predictions, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = 1) + # Add identity line
  labs(x = "Actual Fixed Acidity", y = "Predicted Fixed Acidity", title = "Actual vs Predicted Values for Linear Regression")
```

```
# Display the plot
print(scatter_plot)
```





- `ggplot(predictions, aes(x = Actual, y = Predicted))`: Initializes a ggplot object with the predictions data frame. It maps the actual values (Actual) to the x-axis and the predicted values (Predicted) to the y-axis.
- `geom_point()`: Adds points to the plot, representing the actual versus predicted values.
- `geom_abline(intercept = 0, slope = 1, color = "red", linetype = 1)`: Adds an identity line to the plot. This line has a slope of 1 and passes through the origin (intercept = 0). It helps to visually assess how well the model predictions match the actual values. The color argument sets the color of the line to red, and linetype sets the line type to solid.
- `labs(x = "Actual Fixed Acidity", y = "Predicted Fixed Acidity", title = "Actual vs Predicted Values for Linear Regression")`: Adds labels to the x-axis, y-axis, and a title to the plot.
- As the regression line passes upward sloping inclined towards y-axis this means that predicted fixed acidity is more than the actual one

## # perform ridge regression

```
install.packages("glmnet")
# Load necessary libraries
library(glmnet)
# Subset the training data to include only the columns of interest
train_subset <- train_data[, c("residual.sugar", "chlorides")]
y_train <- train_data$quality
# Perform ridge regression
ridge_model <- glmnet(as.matrix(train_subset), y_train, alpha = 0)
# Summary of the ridge regression model
summary(ridge_model)
```

```
> summary(ridge_model)
      Length Class      Mode
a0         100   -none-   numeric
beta       200 dgCMatrix S4
df          100   -none-   numeric
dim          2   -none-   numeric
lambda      100   -none-   numeric
dev.ratio   100   -none-   numeric
nulldev      1   -none-   numeric
npasses      1   -none-   numeric
jerr          1   -none-   numeric
offset       1   -none-   logical
call         4   -none-    call
nobs         1   -none-   numeric
>
```

- `install.packages("glmnet")`: This line installs the glmnet package if it's not already installed. glmnet provides efficient procedures for fitting the entire lasso or elastic-net regularization path for linear regression, logistic, and multinomial regression models.
- `library(glmnet)`: This line loads the glmnet package, making its functions available for use.
- `train_subset <- train_data[, c("residual.sugar", "chlorides")]`: This line subsets the training data (train\_data) to include only the columns of interest, which are "residual.sugar" and "chlorides". These columns are stored in the train\_subset variable.
- `y_train <- train_data$quality`: This line extracts the response variable ("quality") from the training data and assigns it to the y\_train variable. This will be the variable we're trying to predict using ridge regression.
- `ridge_model <- glmnet(as.matrix(train_subset), y_train, alpha = 0)`: This line fits a ridge regression model using the glmnet function. The predictor variables are provided as a matrix (as.matrix(train\_subset)), and the response variable is y\_train. The alpha = 0 argument specifies that ridge regression should be performed (as opposed to lasso regression, which would be specified by alpha = 1).
- `summary(ridge_model)`: This line provides a summary of the ridge regression model, including information about the coefficients, lambda values, and cross-validated mean squared error.

- *a0*: This is a numeric vector of length 100 containing the intercept term for each value of *lambda* (regularization parameter) in the ridge regression model. In linear regression terms, it represents the intercept term of the regression equation.
- *beta*: This is a dgCMatrx object of class "dgCMatrx" and mode S4. It represents the coefficient matrix for the ridge regression model. In linear regression terms, these coefficients represent the slopes of the predictor variables in the regression equation.
- *df*: This is a numeric vector of length 100 containing the degrees of freedom for each value of *lambda* in the ridge regression model. In linear regression terms, it represents the effective degrees of freedom of the model.
- *dim*: This is a numeric vector of length 2 containing the dimensions of the coefficient matrix. It indicates the number of rows and columns in the coefficient matrix.
- *lambda*: This is a numeric vector of length 100 containing the values of *lambda* (regularization parameter) used in the ridge regression model. In ridge regression, *lambda* controls the amount of shrinkage applied to the coefficients.
- *dev.ratio*: This is a numeric vector of length 100 containing the ratio of deviance explained by the model to the null deviance for each value of *lambda*. In linear regression terms, it represents the proportion of variability explained by the model.
- *nulldev*: This is a numeric scalar representing the null deviance of the model. It measures the variability in the response variable without any predictors.
- *npasses*: This is a numeric scalar indicating the number of passes over the data made during fitting. It may not have a direct interpretation in linear regression terms.
- *jerr*: This is a numeric scalar indicating the error code. It is used internally and may not have a direct interpretation in linear regression terms.
- *offset*: This is a logical scalar indicating whether an offset term was used in the model. An offset term allows you to model the effect of a known quantity on the response variable.
- *call*: This is a call object containing the function call used to fit the ridge regression model.
- *nobs*: This is a numeric scalar representing the number of observations used in fitting the model. It indicates the sample size of the dataset.

### # Predict values using the ridge regression model

```
y_pred_ridge <- predict(ridge_model, s = 0.1, newx = as.matrix(train_subset))
# Create a data frame for actual and predicted values
predictions_ridge <- data.frame(Actual = y_train, Predicted = y_pred_ridge)
# Print the first few rows of the predictions
head(predictions_ridge)
```

```
> # Print the first few rows of the predictions
> head(predictions_ridge)
  Actual      s1
1      5 5.637749
2      5 5.614848
3      5 5.611304
4      6 5.641645
6      5 5.632671
7      5 5.638101
>
```

- *y\_pred\_ridge <- predict(ridge\_model, s = 0.1, newx = as.matrix(train\_subset))*: This line generates predictions (*y\_pred\_ridge*) using the fitted ridge regression model (*ridge\_model*) for the predictor variables in the training data (*train\_subset*). The *s = 0.1* argument specifies the value of *lambda* (regularization parameter) used for prediction, and *newx =*

`as.matrix(train_subset)` converts the predictor variables into matrix format as required by the `predict` function.

- `predictions_ridge <- data.frame(Actual = y_train, Predicted = y_pred_ridge)`: This line creates a data frame named `predictions_ridge` containing the actual values of the response variable (`y_train`) and the predicted values (`y_pred_ridge`) generated by the ridge regression model.
- `head(predictions_ridge)`: This line prints the first few rows of the `predictions_ridge` data frame, allowing you to inspect the actual and predicted values.
- Overall as compared to simple regression prediction ridge regression prediction is more accurate.

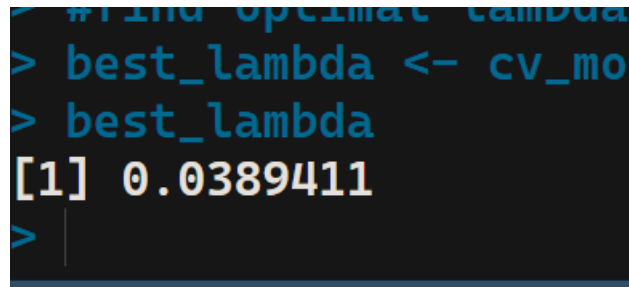
**#perform k-fold cross-validation to find optimal lambda value that minimizes the mean squared error (MSE) on the test data.**

```
cv_model <- cv.glmnet(as.matrix(train_subset), y_train, alpha = 0)
```

```
#find optimal lambda value that minimizes test MSE
```

```
best_lambda <- cv_model$lambda.min
```

```
best_lambda
```

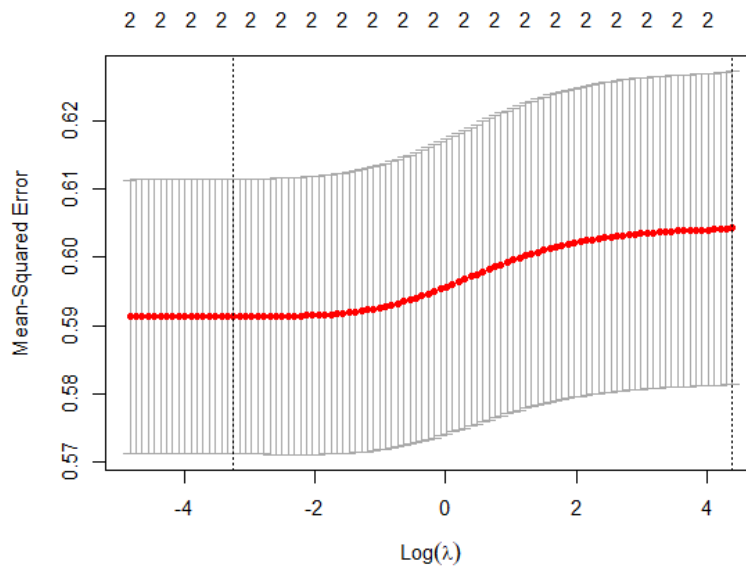


```
#find optimal lambda
> best_lambda <- cv_mod
> best_lambda
[1] 0.0389411
>
```

- `cv_model <- cv.glmnet(as.matrix(train_subset), y_train, alpha = 0)`: This line performs cross-validation using the `cv.glmnet` function. It takes the predictor variables (`as.matrix(train_subset)`), the response variable (`y_train`), and the `alpha` value (`alpha = 0` which indicates ridge regression) as input. The function fits ridge regression models with different values of `lambda`, performs cross-validation to evaluate their performance, and selects the optimal `lambda` value based on the minimum cross-validated mean squared error.
- `best_lambda <- cv_model$lambda.min`: This line extracts the optimal `lambda` value (`lambda.min`) from the cross-validated ridge regression model stored in the `cv_model` object.
- `best_lambda`: This line prints the optimal `lambda` value, which represents the regularization strength that minimizes the mean squared error on the test data.

**#produce plot of test MSE by lambda value**

```
plot(cv_model)
```



- *plot(cv\_model) function generates a plot that visualizes the cross-validated performance of the ridge regression models fitted with different values of lambda (regularization parameter). This plot typically consists of the following components:*
- *Lambda Values on the x-axis: The x-axis of the plot represents the values of lambda (regularization parameter) used in the ridge regression models.*
- *Cross-validated Mean Squared Error (MSE) on the y-axis: The y-axis of the plot represents the cross-validated mean squared error (MSE) for each value of lambda. This metric measures the average squared difference between the predicted and actual values of the response variable across different cross-validation folds.*
- *Three Lines: The plot may include three lines:*
- *Mean MSE Line: This line shows the average cross-validated MSE across different cross-validation folds for each value of lambda. It provides an overall indication of model performance.*
- *1 Standard Error (SE) above Mean MSE: This line represents the mean MSE plus one standard error. It helps in assessing the variability of the cross-validated MSE estimates.*
- *1 Standard Error below Mean MSE: This line represents the mean MSE minus one standard error.*
- *Vertical Dotted Line: This line indicates the value of lambda that minimizes the cross-validated MSE (lambda.min). This lambda value is selected as the optimal regularization parameter based on cross-validation.*
- *Horizontal Dotted Line: This line shows the minimum cross-validated MSE achieved by the model.*

### #find coefficients of best model

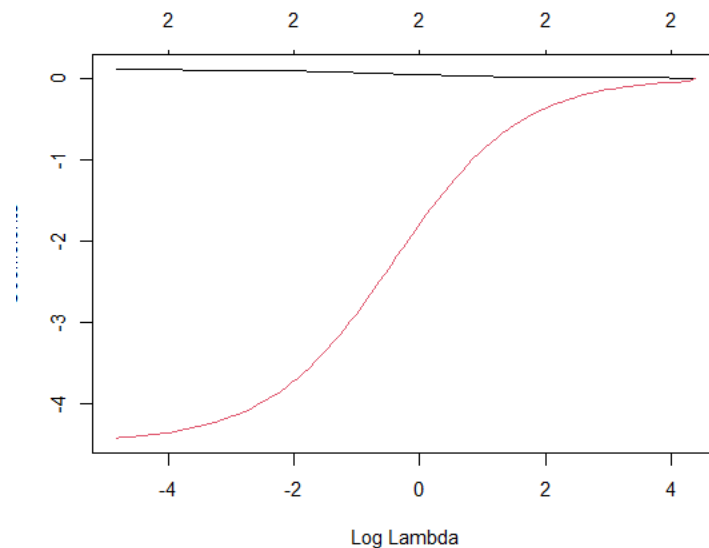
```
best_model <- glmnet( as.matrix(train_subset),y_train,alpha = 0, lambda = best_lambda)
coef(best_model)
```

```
> coef(best_model)
3 x 1 sparse Matrix of class "dgCMatrix"
              s0
(Intercept)  5.77220426
residual.sugar 0.09754684
chlorides    -4.23335423
>
```

- The code you provided fits a ridge regression model (`glmnet`) to the training data using the optimal lambda value (`best_lambda`) obtained from cross-validation.
- `best_model <- glmnet(as.matrix(train_subset), y_train, alpha = 0, lambda = best_lambda)`: This line fits a ridge regression model (`glmnet`) to the training data (`train_subset`) using the specified alpha value (0 for ridge regression) and the optimal lambda value (`best_lambda`). The `glmnet` function uses the specified alpha value and lambda value to fit the ridge regression model.
- `coef(best_model)`: This line retrieves the coefficients of the fitted ridge regression model stored in the `best_model` object. It returns a matrix containing the coefficients for each predictor variable.
- The coefficients represent the slopes of the predictor variables in the ridge regression equation. These coefficients are shrunk towards zero due to the regularization effect of ridge regression. Positive coefficients indicate a positive relationship with the response variable, while negative coefficients indicate a negative relationship. The magnitude of the coefficients reflects their importance in predicting the response variable.

### #produce Ridge trace plot

`plot(ridge_model, xvar = "lambda")`



- `plot()` function with a `ridge_model` object as input generates a plot showing the coefficients' paths as a function of the regularization parameter (`lambda`) for ridge regression. However, specifying `xvar = "lambda"` explicitly sets the x-axis to represent the lambda values.
- **x-axis (lambda values)**: The x-axis represents the values of the regularization parameter `lambda` used in the ridge regression. Each point on the plot corresponds to a different lambda value.
- **y-axis (coefficients)**: The y-axis represents the coefficients of the predictor variables. Each coefficient's path (trajectory) is plotted as a function of lambda.
- **Lines**: Each line on the plot corresponds to a different predictor variable. As lambda increases (moving right on the x-axis), the coefficients tend to shrink towards zero due to the penalty term added to the objective function in ridge regression.
- By visualizing the coefficient paths as a function of lambda, you can observe how the coefficients change in magnitude and direction as the regularization strength varies. This can help in understanding the effect of regularization on the coefficients and identifying which predictors are more important in the model.

### # Find SST and SSE and R-squared of predicted model

```
x<-data.matrix(train_data[,c("residual.sugar", "chlorides")])
#use fitted best model to make predictions
```

```

y_predicted <- predict(ridge_model, s = best_lambda, newx = x)
#find SST and SSE
sst <- sum((y - mean(y_train))^2)
sse <- sum((y_predicted - y_train)^2)
#find R-Squared
rsq <- 1 - sse/sst
rsq

```

```

> rsq <- 1 - sse/sst
> rsq
[1] 0.9700204
>

```

- `x <- data.matrix(train_data[,c("residual.sugar", "chlorides")])`: This line extracts the predictor variables "residual.sugar" and "chlorides" from the training data (train\_data) and converts them into a matrix format (data.matrix). This matrix is stored in the variable x.
- `y_predicted <- predict(ridge_model, s = best_lambda, newx = x)`: This line uses the fitted ridge regression model (ridge\_model) to make predictions (y\_predicted) for the response variable using the predictor variables x. The `s = best_lambda` argument specifies the optimal lambda value obtained from cross-validation. The `newx = x` argument provides the predictor variables in matrix format.
- `sst <- sum((y - mean(y_train))^2)`: This line calculates the total sum of squares (SST), which represents the total variability in the response variable (y\_train). It computes the squared differences between each observed response variable (y\_train) and the mean of the response variable.
- `sse <- sum((y_predicted - y_train)^2)`: This line calculates the residual sum of squares (SSE), which represents the unexplained variability in the response variable by the ridge regression model. It computes the squared differences between each predicted response variable (y\_predicted) and the observed response variable (y\_train).
- `rsq <- 1 - sse/sst`: This line calculates the coefficient of determination (R-squared) by subtracting the ratio of SSE to SST from 1. R-squared measures the proportion of the total variability in the response variable that is explained by the predictor variables in the model.
- `rsq`: This line prints the calculated R-squared value, which indicates the goodness of fit of the ridge regression model to the training data. Higher R-squared values indicate better fit, with 1 representing a perfect fit. As value is 0.97 it is best fit.

**# one-sample t-test to assess whether the mean of the "sulphates" variable in the "cleaned\_red\_wine\_data" dataset is significantly different from a specified population mean of 0.55.**

```
t.test(cleaned_red_wine_data$sulphates,mu=0.55)
```

```

One Sample t-test

data:  cleaned_red_wine_data$sulphates
t = 27.17, df = 1450, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0.55
95 percent confidence interval:
 0.6359001 0.6492687
sample estimates:
mean of x
0.6425844

```

**Hypothesis:**

- **Null Hypothesis (H0):** The mean of the "sulphates" variable in the population is equal to 0.55.



- *Alternative Hypothesis (H1): The mean of the "sulphates" variable in the population is not equal to 0.55.*
- *Test Statistic: The t-test uses the t-statistic to compare the sample mean to the population mean.*

#### **Assumptions:**

- *The data are independent and identically distributed.*
- *The data are approximately normally distributed.*
- *The population standard deviation is unknown.*

*One Sample t-test: This indicates that a one-sample t-test was performed.*

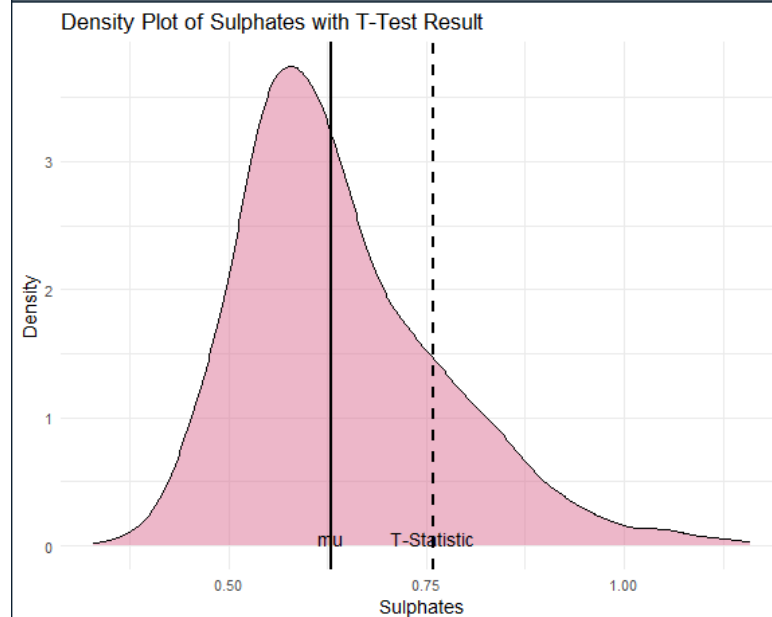
- *data: It specifies the data on which the t-test was conducted, which is the "sulphates" variable in the "cleaned\_red\_wine\_data" dataset.*
- *t: This is the value of the t-statistic calculated for the test. In this case, the calculated t-value is approximately 0.75844.*
- *df: This represents the degrees of freedom associated with the t-test. Here, it's reported as 1450.*
- *p-value: This is the probability associated with the observed t-value under the null hypothesis. In this case, the p-value is approximately 0.4483.*
- *alternative hypothesis: It specifies the alternative hypothesis being tested. Here, it states that the true mean of the "sulphates" variable is not equal to 0.64.*
- *95 percent confidence interval: This provides the confidence interval estimate for the true mean of the "sulphates" variable. It ranges from approximately 0.6359 to 0.6493.*
- *sample estimates: This section provides the sample estimate of the mean of the "sulphates" variable, which is approximately 0.6426.*

#### **Interpretation:**

- *Given the p-value of 0.4483, which is greater than the significance level of 0.05 (assuming a standard significance level), we fail to reject the null hypothesis. Therefore, we do not have enough evidence to conclude that the true mean of the "sulphates" variable is significantly different from 0.64. Additionally, the confidence interval for the mean provides a range of plausible values for the true mean of the "sulphates" variable.*

### **# Create a density plot of the one-sample t-test result**

```
density_plot <- ggplot(cleaned_red_wine_data, aes(x = sulphates)) +
  geom_density(fill = "palevioletred", alpha = 0.5) +
  geom_vline(xintercept = 0.63, color = "black", lty = 1, size = 1) +
  annotate(geom = "text", x = 0.63, y = 0, label = "mu", vjust = 0) + # Add label for t-statistic+
  geom_vline(xintercept = t_test_result$statistic, color = "red", linetype = "dashed", size = 1) +
  # Add vertical line for t-statistic
  annotate(geom = "text", x = t_test_result$statistic, y = 0, label = "T-Statistic", vjust = 0) + # Add label
  for t-statistic
  labs(x = "Sulphates", y = "Density", title = "Density Plot of Sulphates with T-Test Result") +
  theme_minimal()
# Print the density plot
print(density_plot)
```



- *ggplot():* Initializes a ggplot object with the "cleaned\_red\_wine\_data" dataset and specifies the aesthetic mappings.
- *geom\_density():* Adds a density plot layer to the ggplot object, representing the distribution of the "sulphates" variable. The fill color is set to "palevioletred" with an alpha value of 0.5 for transparency.
- *geom\_vline():* Adds vertical lines to the plot.
- The first *geom\_vline()* call adds a vertical line at  $x = 0.63$ , representing the specified value of  $\mu$  (population mean).
- The second *geom\_vline()* call adds a vertical line at the t-statistic value obtained from a t-test result (`t_test_result$statistic`). The color of this line is set to red, and it's displayed with a dashed line type.
- *annotate():* Adds text annotations to the plot.
- The first *annotate()* call adds a text label "mu" at  $x = 0.63$ .
- The second *annotate()* call adds a text label "T-Statistic" at the position corresponding to the t-statistic value.
- *labs():* Sets the labels for the x-axis, y-axis, and plot title.
- *theme\_minimal():* Applies the minimal theme to the plot.
- *print():* Displays the density plot.

**# perform a simple linear regression analysis using the `lm()` function in R, where the response variable is "total.sulfur.dioxide" and the predictor variable is "free.sulfur.dioxide".**

```
regressor1=lm(formula = total.sulfur.dioxide~free.sulfur.dioxide,data=train_data)
summary(regressor1)
```

```

Call:
lm(formula = total.sulfur.dioxide ~ free.sulfur.dioxide, data = train_
a)

Residuals:
    Min       1Q   Median       3Q      Max
-47.304 -12.861  -6.583   7.155  99.479

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    12.53092    1.29586   9.67  <2e-16 ***
free.sulfur.dioxide  2.06603    0.07217  28.63  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 22.3 on 1086 degrees of freedom
Multiple R-squared:  0.4301,    Adjusted R-squared:  0.4295
F-statistic: 819.5 on 1 and 1086 DF,  p-value: < 2.2e-16

```

- *regressor1*: This variable stores the result of the linear regression model.
- *lm()*: This function is used to fit a linear regression model. The formula `total.sulfur.dioxide ~ free.sulfur.dioxide` specifies that "total.sulfur.dioxide" is the response variable, and "free.sulfur.dioxide" is the predictor variable.
- *data*: The data argument specifies the dataset (`train_data`) containing the variables used in the regression analysis.
- *summary(regressor1)*: This function provides a summary of the fitted regression model `regressor1`. The summary includes various statistics such as coefficients, standard errors, t-values, p-values, and R-squared.

*Residuals*: Describes the distribution of the residuals (the differences between observed and predicted values).

- *Min*: The minimum residual is approximately -47.304.
- *1Q, Median, 3Q*: These are the first quartile, median, and third quartile of the residuals.
- *Max*: The maximum residual is approximately 99.479.
- *Coefficients*: Provides estimates for the intercept and slope of the regression line.
- *Intercept*: The intercept is approximately 12.53092.
- *free.sulfur.dioxide*: The coefficient for "free.sulfur.dioxide" is approximately 2.06603. This means that for each unit increase in "free.sulfur.dioxide", the "total.sulfur.dioxide" is expected to increase by approximately 2.06603 units.

*Significance Codes*: Indicates the level of significance of each coefficient.

- The number of asterisks (\*) indicates the level of significance. Here, both coefficients are highly significant, as indicated by the three asterisks (\*\*\*), with p-values much smaller than 0.05.
- *Residual Standard Error*: Represents the standard deviation of the residuals, approximately 22.3.
- *Multiple R-squared*: Indicates the proportion of variance in the response variable ("total.sulfur.dioxide") explained by the predictor variable ("free.sulfur.dioxide"). Here, it's approximately 0.4301, suggesting that around 43.01% of the variability in "total.sulfur.dioxide" is explained by "free.sulfur.dioxide".
- *Adjusted R-squared*: Similar to R-squared but adjusted for the number of predictors in the model.

*F-statistic*: Tests the overall significance of the model.

- The F-statistic is 819.5, with a very low p-value (< 2.2e-16), indicating that the model as a whole is statistically significant.

Overall, this output provides information about the fit of the linear regression model, the significance of the coefficients, and the overall explanatory power of the model in predicting "total.sulfur.dioxide" based on "free.sulfur.dioxide".

**# conduct predictions using the linear regression model `regressor1` on the test data, and then it compares the predicted values with the actual values.**

```

y_prediction=predict(regressor1,newdata = test_data)
y_prediction
predictions1 <- data.frame(Actual = test_data$total.sulfur.dioxide, Predicted = y_prediction)
head(predictions1)

```

```

> head(predictions1)
  Actual Predicted
5      34   35.25723
8      21   43.52134
11     65   43.52134
22     71   60.04957
25     40   55.91751
28     37   33.19120

```

- *y\_prediction <- predict(regressor1, newdata = test\_data)* This line calculates the predicted values of the response variable ("total.sulfur.dioxide") using the linear regression model regressor1. The predict() function is applied to the fitted regression model (regressor1) with the argument newdata = test\_data, which specifies the new dataset (test\_data) on which predictions are to be made. The predicted values are stored in the variable y\_prediction.
- *y\_prediction* This line simply prints out the predicted values (y\_prediction). It displays the predicted values of "total.sulfur.dioxide" for each observation in the test dataset.
- *predictions1 <- data.frame(Actual = test\_data\$total.sulfur.dioxide, Predicted = y\_prediction)* This line creates a data frame named predictions1. It contains two columns:
- *Actual:* Holds the actual values of "total.sulfur.dioxide" from the test dataset (test\_data\$total.sulfur.dioxide).
- *Predicted:* Contains the predicted values of "total.sulfur.dioxide" generated by the linear regression model (y\_prediction).

**# conducts a paired t-test to compare two related samples, specifically the differences between the predicted and actual values of "total.sulfur.dioxide"**

```

before<-predictions1$Actual
after<-predictions1$Predicted
t.test(x=before,y=after,paired=TRUE)

```

```

Paired t-test

data:  before and after
t = -0.0085145, df = 362, p-value = 0.9932
alternative hypothesis: true mean difference is not equal to 0
95 percent confidence interval:
 -2.239000  2.219695
sample estimates:
mean difference
 -0.00965239
>

```

- **before:** This variable represents the actual values of "total.sulfur.dioxide" from the predictions1 data frame.
- **after:** This variable represents the predicted values of "total.sulfur.dioxide" from the predictions1 data frame.
- **t.test():** This function conducts a paired t-test on the two related samples.
- **x:** Represents the values before an intervention (in this case, the actual values).
- **y:** Represents the values after an intervention (in this case, the predicted values).
- **paired = TRUE:** Specifies that the samples are paired.

- The paired t-test evaluates whether there is a significant difference between the means of the two related samples. In this context, it assesses whether there is a significant difference between the actual and predicted values of "total.sulfur.dioxide".

#### Paired t-test:

- **data:** Indicates that the t-test was conducted on the "before" and "after" samples.
- **t:** The calculated t-value is approximately -0.0085145.
- **df:** The degrees of freedom associated with the t-distribution is reported as 362.
- **p-value:** The p-value associated with the t-test is approximately 0.9932.
- **alternative hypothesis:** States that the true mean difference between the "before" and "after" samples is not equal to zero.
- **95 percent confidence interval:** Provides a confidence interval for the true mean difference, ranging from -2.239000 to 2.219695.
- **sample estimates:** The mean difference between the "before" and "after" samples is estimated to be approximately -0.00965239.

#### Interpretation:

- Since the p-value (0.9932) is much greater than the typical significance level of 0.05, we fail to reject the null hypothesis. This indicates that there is no significant difference between the mean values of the "before" and "after" samples. In other words, the actual and predicted values of "total.sulfur.dioxide" are not significantly different from each other. Additionally, the confidence interval for the mean difference includes zero, further supporting this conclusion. Therefore, based on this analysis, there is no evidence to suggest that the predictive model significantly differs from the actual values.

**# calculate the differences between the "after" (predicted) and "before" (actual) values of "total.sulfur.dioxide".**

**differences <- after – before**

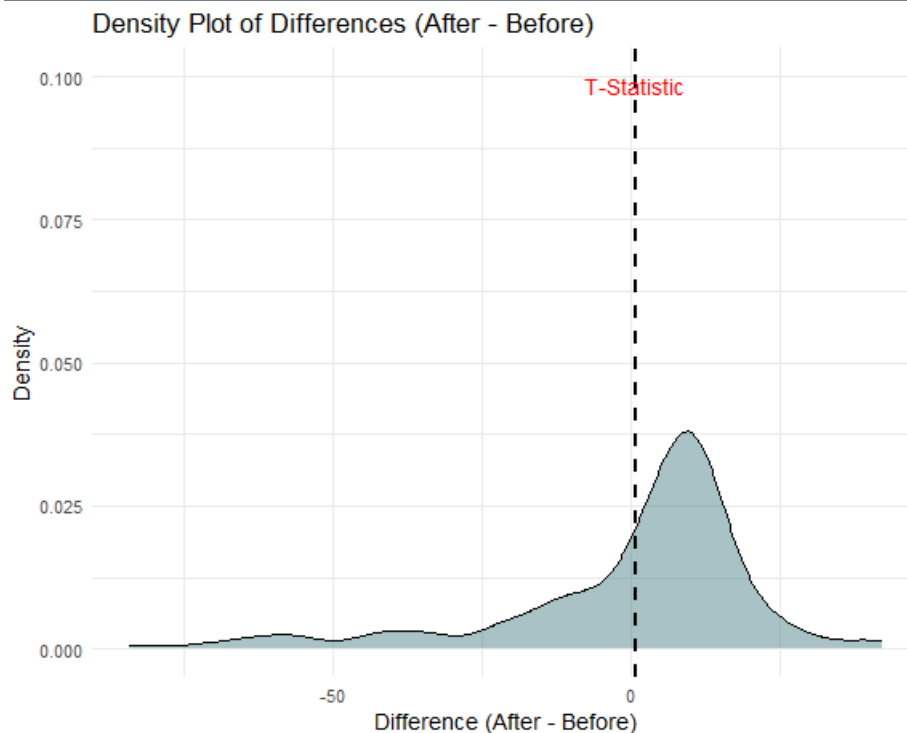
- *differences: This variable represents the differences between the predicted ("after") and actual ("before") values of "total.sulfur.dioxide". It subtracts the "before" values from the "after" values for each corresponding observation.*
- *By calculating these differences, you can further analyze the extent and direction of the discrepancies between the predicted and actual values. Positive differences indicate that the predicted values are higher than the actual values, while negative differences indicate that the predicted values are lower than the actual values.*

**# Create density plot of the differences**

```
density_plot <- ggplot(data.frame(Differences = differences), aes(x = Differences)) +
  geom_density(fill = "cadetblue4", alpha = 0.5) +
  labs(x = "Difference (After - Before)", y = "Density", title = "Density Plot of Differences (After - Before)") +
  geom_vline(xintercept = t_test_result$statistic, color = "black", linetype = "dashed", size=1) + # Add vertical line for t-statistic
  annotate(geom = "text", x = t_test_result$statistic, y = 0.1, label = "T-Statistic", color = "red", vjust = 1) + # Add label for t-statistic
  theme_minimal()
```

**# Print the density plot**

```
print(density_plot)
```



- *density\_plot*: This variable stores the *ggplot* object, which represents the density plot of the differences between the predicted and actual values of "total.sulfur.dioxide".
- *ggplot()*: Initializes a *ggplot* object with the differences data frame as the data source.
- *geom\_density()*: Adds a density plot layer to the *ggplot* object, representing the distribution of the differences. The fill color is set to "cadetblue4" with an alpha value of 0.5 for transparency.
- *labs()*: Sets the labels for the x-axis, y-axis, and plot title.
- *geom\_vline()*: Adds a vertical line to the plot, representing the t-statistic value obtained from a t-test result.
- *annotate()*: Adds a text annotation to the plot, labeling the t-statistic.
- *theme\_minimal()*: Applies the minimal theme to the plot.
- *print()*: Displays the density plot.
- Overall, this code segment generates a density plot of the differences between the predicted and actual values of "total.sulfur.dioxide", providing insights into the distribution and variability of the prediction errors. Additionally, it annotates the plot with the t-statistic value obtained from a t-test result, aiding in the interpretation of the plot in the context of statistical hypothesis testing.

### #fit one-way ANOVA model

```
model <- aov(density ~ citric.acid, data = test_data)
```

```
summary(model)
```

```
> summary(model)
          Df    Sum Sq   Mean Sq  F value    Pr(>F)
citric.acid  1 0.0001921 1.921e-04    77.98 <2e-16 ***
Residuals 361 0.0008891 2.460e-06
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
>
```

- *aov()*: This function performs an analysis of variance (ANOVA) to evaluate the impact of one or more predictor variables on a continuous response variable.
- *density ~ citric.acid*: Specifies the model formula, indicating that "density" is the response variable and "citric.acid" is the predictor variable.
- *data = test\_data*: Specifies the dataset (*test\_data*) from which the variables are taken.
- *summary()*: This function provides a summary of the ANOVA model.

- *model: The ANOVA model object generated by the aov() function.*

#### **Df (Degrees of Freedom):**

- *For "citric.acid": 1 degree of freedom, indicating the number of levels or groups within the variable "citric.acid".*
- *For "Residuals": 361 degrees of freedom, representing the leftover degrees of freedom after accounting for the model.*
- *Sum Sq (Sum of Squares):*
- *For "citric.acid": 0.0001921, which represents the sum of squares associated with the variability explained by the predictor variable "citric.acid".*
- *For "Residuals": 0.0008891, which represents the sum of squares of the residuals, i.e., the unexplained variability in the model.*
- *Mean Sq (Mean Square):*
- *For "citric.acid": 1.921e-04, which is the sum of squares divided by the degrees of freedom for "citric.acid".*
- *For "Residuals": 2.460e-06, which is the sum of squares of the residuals divided by the degrees of freedom for residuals.*

#### **F value:**

- *For "citric.acid": 77.98, which is the ratio of the mean square of "citric.acid" to the mean square of the residuals. It assesses the overall significance of the predictor variable "citric.acid".*
- *Pr(>F) (p-value):*
- *For "citric.acid": <2e-16 (very close to zero), indicating a highly significant p-value.*
- *This small p-value suggests strong evidence against the null hypothesis, indicating that the predictor variable "citric.acid" has a significant effect on the response variable "density".*

#### **Significance Codes:**

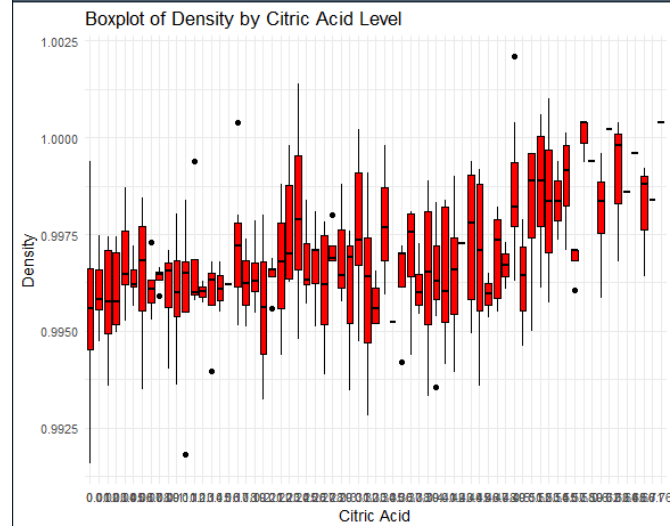
- *The significance codes provide a visual representation of the significance level of the predictors. In this case, "\*\*\*\*" indicates extremely high significance.*
- *Interpretation:*
- *The results indicate that the predictor variable "citric.acid" has a highly significant effect on the response variable "density", as evidenced by the very low p-value (<2e-16) and the significant F-value (77.98). This suggests that there is strong evidence to reject the null hypothesis, indicating that "citric.acid" is associated with significant changes in "density".*

**# create a boxplot to visualize the distribution of the response variable ("density") across different levels of the predictor variable ("citric.acid").**

```
boxplot <- ggplot(model, aes(x = factor(citric.acid), y = density)) +
  geom_boxplot(fill = "red", color = "black") +
  labs(x = "Citric Acid", y = "Density", title = "Boxplot of Density by Citric Acid Level") +
  theme_minimal()
```

```
# Print the boxplot
print(boxplot)
```





- ***ggplot():*** Initializes a ggplot object.
- ***aes():*** Specifies aesthetics mappings. *x = factor(citric.acid)* maps the levels of the predictor variable "citric.acid" to the x-axis, and *y = density* maps the response variable "density" to the y-axis.
- ***geom\_boxplot():*** Adds a boxplot layer to the ggplot object, displaying the distribution of "density" across different levels of "citric.acid". The fill color is set to "red" and the outline color to "black".
- ***labs():*** Sets the axis labels and plot title.
- ***theme\_minimal():*** Applies the minimal theme to the plot, removing unnecessary elements.
- ***print():*** Displays the boxplot.
- This boxplot allows you to visually compare the distribution of "density" across different levels of "citric.acid". It provides insights into the central tendency, spread, and variability of "density" within each level of "citric.acid".