1) Linear Regression

2023-03-19

Pre-processing the data-set

```
data <- read.csv("Linear_Regression_Dataset_new.csv", header = TRUE)
processed_data <- na.omit(data)
head(processed_data)
```

```
##    x        y
## 1 24 21.54945
## 2 50 47.46446
## 3 15 17.21866
## 4 38 36.58640
## 5 87 87.28898
## 6 36 32.46387
```

```
summary(processed_data)
```

```
##        x                y
##  Min.   :  0.00   Min.   : -3.84
##  1st Qu.: 25.00   1st Qu.: 25.19
##  Median : 50.00   Median : 49.93
##  Mean   : 50.29   Mean   : 50.32
##  3rd Qu.: 74.50   3rd Qu.: 74.48
##  Max.   :100.00   Max.   :108.87
```

```
str(processed_data)
```

```
## 'data.frame':    999 obs. of  2 variables:
##  $ x: num  24 50 15 38 87 36 12 81 25 5 ...
##  $ y: num  21.5 47.5 17.2 36.6 87.3 ...
##  - attr(*, "na.action")= 'omit' Named int 214
##   ..- attr(*, "names")= chr "214"
```

```
nrow(processed_data)
```

```
## [1] 999
```

Splitting the model

```
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
indexs = createDataPartition(processed_data$y, times = 1, p = 0.7, list = F)
#times = no. of times to be split
#p = percentage of data to be used for training, here 70% is used of training and 30%
for testing

train = processed_data[indexs, ]
nrow(train)
```

```
## [1] 700
```

```
test = processed_data[-indexs, ]
nrow(test)
```

```
## [1] 299
```

Creating the model

```
cor(train$x, train$y)
```

```
## [1] 0.995206
```

```
#y = dependent
#x = independent
#y = slope * x + intercept
# dependent ~ independent
model <- lm(y ~ x, data = train)
model
```

```
##
## Call:
## lm(formula = y ~ x, data = train)
##
## Coefficients:
## (Intercept)            x
##     -0.2486       1.0058
```

```
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x, data = train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.5202 -1.8853 -0.0699  1.8715  8.1150
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.248633   0.215237  -1.155    0.248
## x            1.005801   0.003741 268.842   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.816 on 698 degrees of freedom
## Multiple R-squared:  0.9904, Adjusted R-squared:  0.9904
## F-statistic: 7.228e+04 on 1 and 698 DF,  p-value: < 2.2e-16
```

Predicting the values using the model

```
#df <- data.frame(x = c(29)), just to initially check
predicted <- predict(model, test)
predicted
```

```
##          5          10          13          16          17          18 
## 87.2560222   4.7803703  23.8905824  60.0994051  25.9021836  73.1748134 
##         21          22          23          25          35          45 
## 68.1458102  87.2560222  58.0878039  84.2386203  60.0994051  35.9601900 
##         49          50          54          61          62          68 
## 18.8615792  32.9427881  59.0936045  58.0878039  74.1806140  -0.2486328 
##         69          72          73          74          75          79 
## 11.8209748  64.1226077   4.7803703  58.0878039  31.9369874   4.7803703 
##         82          90          93          96         115         116 
##  2.7687691  95.3024273  41.9949938  55.0704020  40.9891931  26.9079843 
##        117         120         121         126         127         129 
## 58.0878039  70.1574115  71.1632121  45.0123957  40.9891931  36.9659906 
##        132         136         141         142         146         147 
## 47.0239969  95.3024273  56.0762026  80.2154178  95.3024273  94.2966267 
##        149         152         157         158         166         168 
##  6.7919716  48.0297976  25.9021836  51.0471995  73.1748134  80.2154178 
##        169         174         175         177         183         189 
## 77.1980159  56.0762026  47.0239969   1.7629684  40.9891931  16.8499779 
##        196         200         205         207         209         212 
## 71.1632121  51.0471995   2.7687691  23.8905824  58.0878039  13.8325760 
##        217         222         228         232         236         241 
## 71.1632121  74.1806140  71.1632121  99.3256298  65.1284083  28.9195855 
##        243         246         248         249         255         256 
##  6.7919716  50.0413988  89.2676235  76.1922153  66.1342089  50.0413988 
##        258         259         270         277         283         286 
## 60.0994051  34.9543893 100.3314305  37.9717912  44.0065950   4.7803703 
##        293         294         301         302         303         305 
## 55.0704020   3.7745697  75.1864146  33.9485887  37.9717912  20.8731805 
##        306         309         310         312         313         320 
## 88.2618229  44.0065950   9.8093735  15.8441773  31.9369874  34.9543893 
##        321         322         326         327         329         332 
## 23.8905824  16.8499779  89.2676235  69.1516108   5.7861710   8.8035729 
##        334         335         343         346         349         354 
## 50.0413988  85.2444210  18.8615792  58.0878039  19.8673798  98.3198292 
##        355         357         358         360         363         367 
## 90.2734241  93.2908260  21.8789811  12.8267754  30.9311868  22.8847817 
##        369         372         374         383         387         390 
## 88.2618229  91.2792248   9.8093735  26.9079843  46.0181963   8.8035729 
##        393         400         406         407         417         418 
##  0.7571678  91.2792248  23.8905824  48.0297976  41.9949938  95.3024273 
##        425         431         432         433         434         437 
## 25.9021836  23.8905824  70.1574115  28.9195855  76.1922153  87.2560222 
##        438         442         443         446         449         451 
##  8.8035729  16.8499779  49.0355982  89.2676235  47.0239969   4.7803703 
##        452         455         467         472         473         481 
## 68.1458102  40.9891931  50.0413988  18.8615792  94.2966267  50.0413988 
##        483         486         490         492         494         499 
## 29.9253862  15.8441773  62.1110064  29.9253862  62.1110064  20.8731805 
##        501         504         505         510         518         520 
## 97.3140286  47.0239969  98.3198292  17.8557786  91.2792248  85.2444210 
##        527         532         534         540         542         546 
## 36.9659906  68.1458102   4.7803703  25.9021836  60.0994051  28.9195855 
##        557         558         562         563         564         566 
## 20.8731805  98.3198292  30.9311868  93.2908260  67.1400096  24.8963830 
##        569         577         588         591         592         594 
```

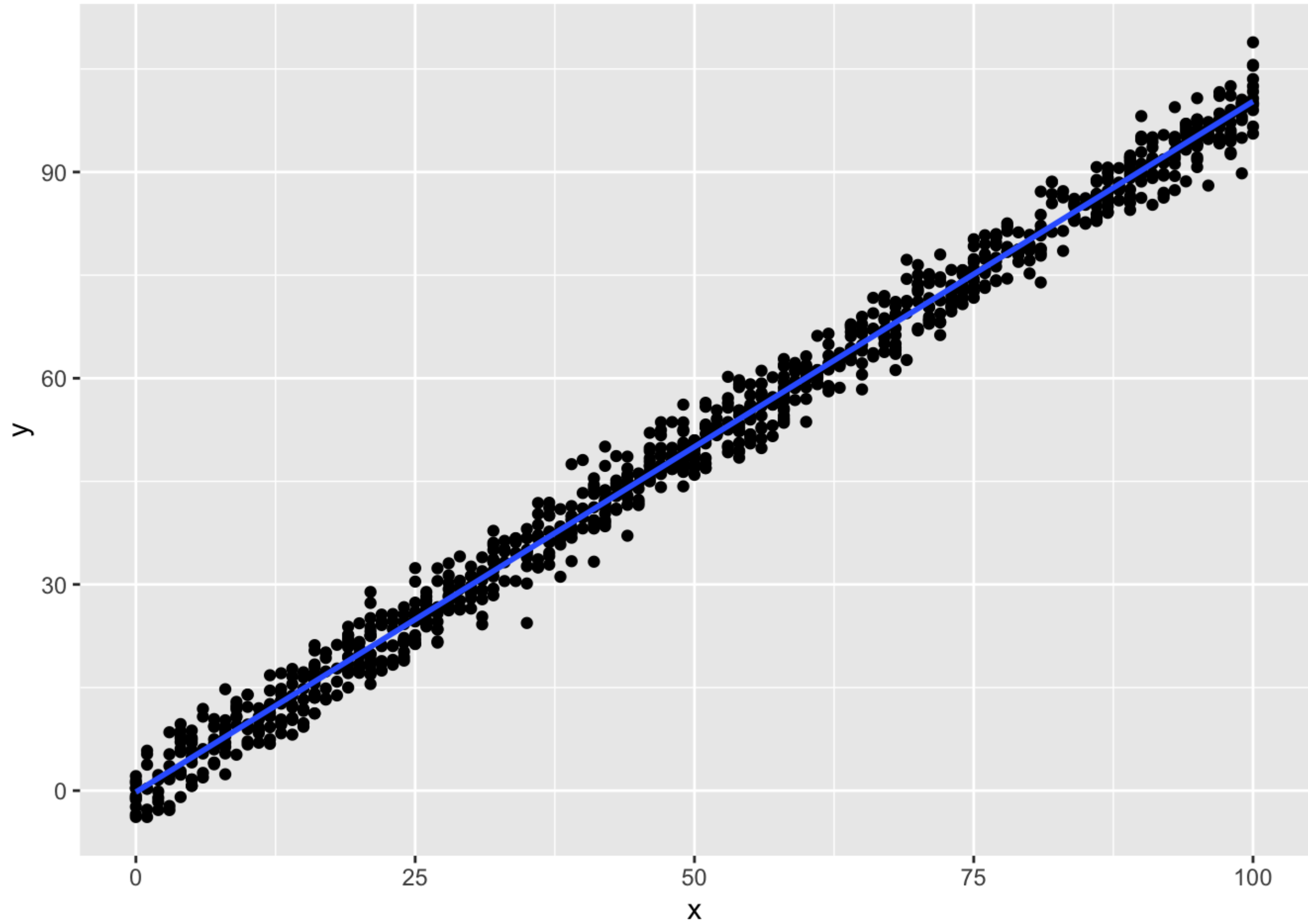
```
##  20.8731805  67.1400096  49.0355982  70.1574115  91.2792248  54.0646014
##         595         596         598         599         601         617
##  38.9775919  91.2792248  21.8789811   1.7629684  65.1284083  49.0355982
##         619         621         622         623         625         627
##  46.0181963  89.2676235  36.9659906  28.9195855  96.3082279  74.1806140
##         628         631         632         633         634         637
##  34.9543893  56.0762026  17.8557786 100.3314305  54.0646014  81.2212184
##         640         642         643         645         646         649
##   0.7571678  13.8325760  24.8963830  98.3198292  97.3140286  88.2618229
##         660         663         665         668         674         678
##   3.7745697  34.9543893   9.8093735  58.0878039  61.1052058  43.0007944
##         681         683         687         696         705         709
##  45.0123957  33.9485887  67.1400096  58.0878039  35.9601900  19.8673798
##         710         714         718         723         729         731
##   4.7803703  62.1110064  13.8325760  89.2676235  93.2908260  99.3256298
##         738         739         744         752         754         757
##  69.1516108  27.9137849  76.1922153  88.2618229  30.9311868  38.9775919
##         758         761         762         768         770         776
##  64.1226077  12.8267754  72.1690127  50.0413988  12.8267754  27.9137849
##         777         781         782         791         793         794
##  81.2212184  68.1458102  26.9079843  67.1400096  63.1168070  91.2792248
##         796         801         805         810         811         812
##  13.8325760   1.7629684  41.9949938  -0.2486328  40.9891931  15.8441773
##         813         815         816         817         818         820
##  94.2966267  66.1342089  23.8905824  16.8499779  90.2734241  -0.2486328
##         821         823         827         829         830         833
##  64.1226077  98.3198292  78.2038165  89.2676235  28.9195855  11.8209748
##         835         843         847         852         853         858
##  27.9137849  73.1748134  99.3256298  31.9369874  94.2966267  93.2908260
##         862         866         867         870         876         880
##  46.0181963  43.0007944  95.3024273  34.9543893  31.9369874  72.1690127
##         884         892         895         896         898         901
##  91.2792248  73.1748134  61.1052058  99.3256298  72.1690127  78.2038165
##         912         913         915         918         920         925
##  45.0123957  59.0936045  22.8847817  95.3024273   3.7745697  96.3082279
##         927         928         929         930         933         935
## 100.3314305  87.2560222  13.8325760  13.8325760  88.2618229  65.1284083
##         938         939         943         945         946         947
##  15.8441773   4.7803703  54.0646014  44.0065950  30.9311868  68.1458102
##         949         952         957         963         964         965
##  90.2734241  95.3024273  89.2676235  45.0123957  73.1748134  57.0820032
##         966         969         971         974         977         983
##  19.8673798  55.0704020  55.0704020  72.1690127  96.3082279  64.1226077
##         988         989         991         995         999
##  41.9949938  43.0007944  92.2850254   7.7977722  62.1110064
```

Plotting the linear regression curve

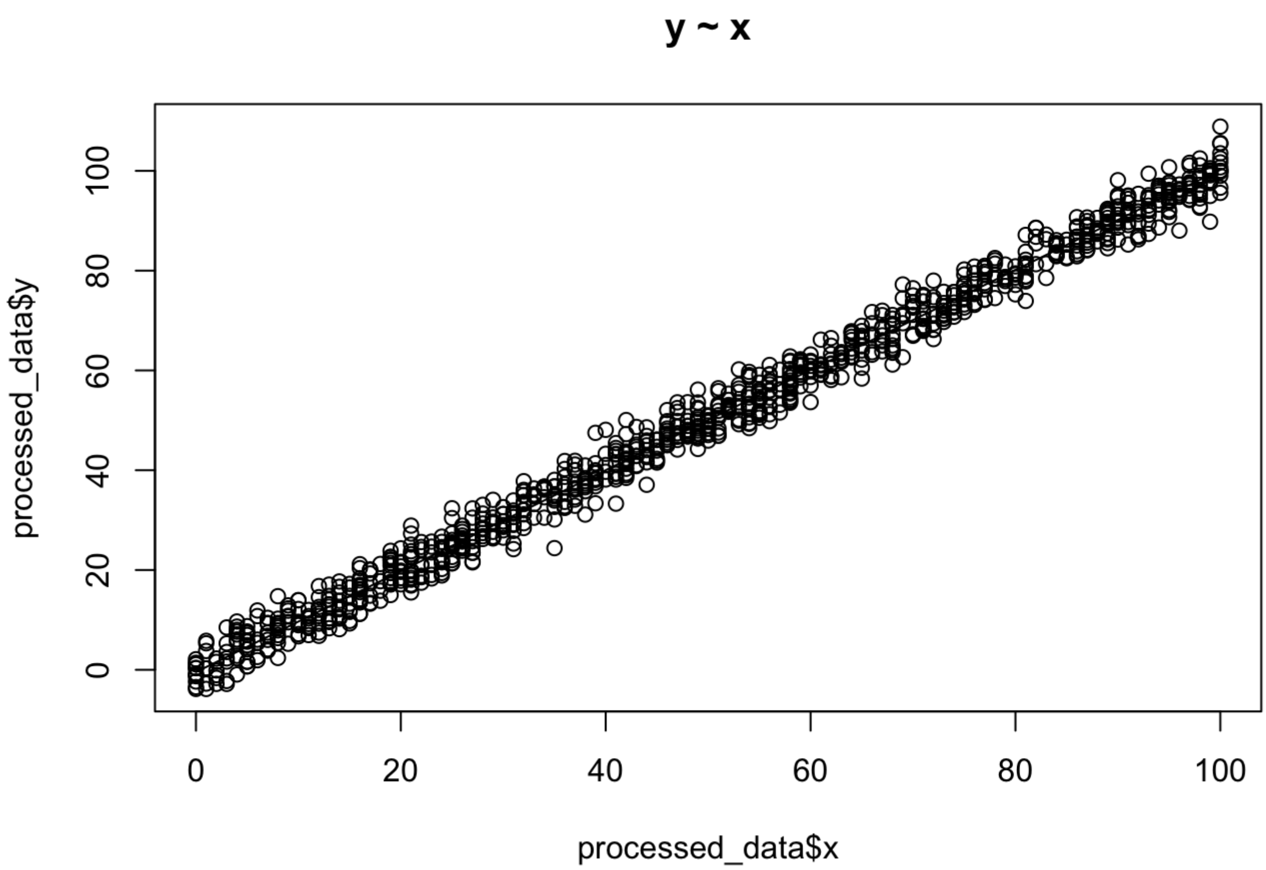
```
library(ggplot2)

ggplot(processed_data, aes(x = x, y = y)) + geom_point() + geom_smooth(method = 'lm')
```

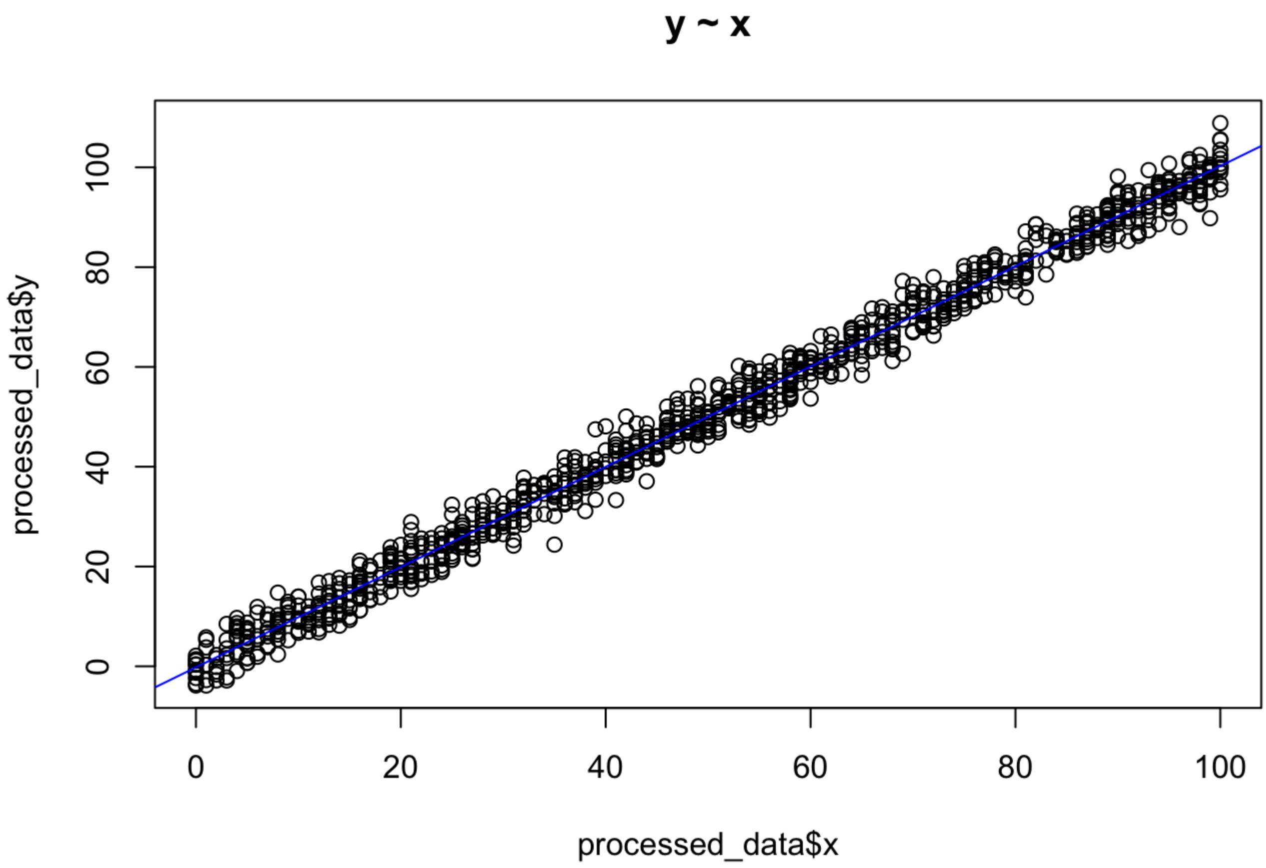
```
## `geom_smooth()` using formula = 'y ~ x'
```



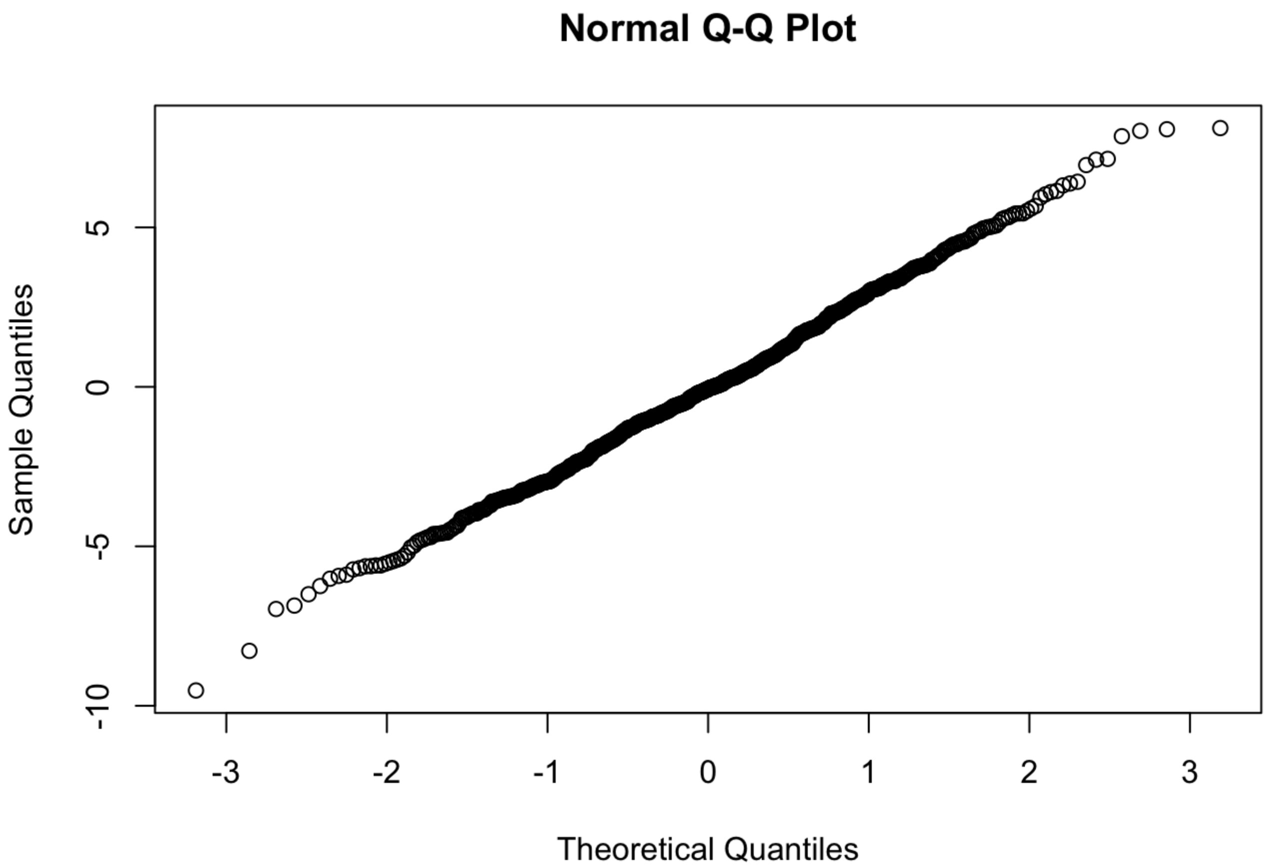

```
scatter.smooth(x = processed_data$x, y = processed_data$y, main = "y ~ x")
```



```
plot(processed_data$x, processed_data$y, main = "y ~ x")
abline(lm(processed_data$y ~ processed_data$x, data = processed_data), col = "blue")
```

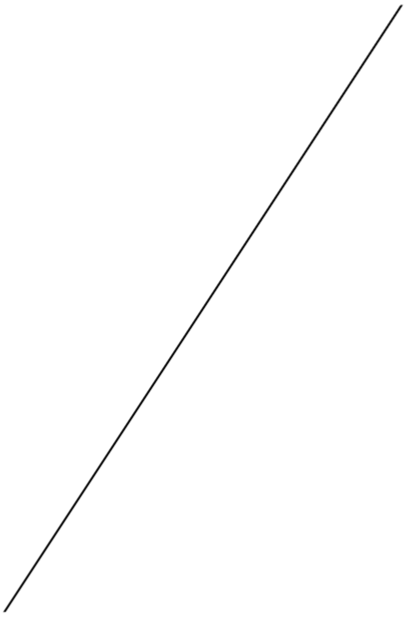



```
res <- model$residuals
#create Q-Q plot for residuals
qqnorm(res)
```




```
res1 <- model$residuals

plot.new()
#text(x = 0.5, y = 0.5, labels = "This is a new plot canvas.")
qqline(res1)
```



Confusion Matrix: It is not used in Linear Regression as Confusion Matrix is only applied to classification problems, not regression problems.

Conclusion: As we can see, the model is performing poorly as the dataset is containing less number of numerical data and fewer relationships among them. As we can observe in the correlation coefficient calculation, the relationship between age and charges is not more than 50%.To improve the model, we can convert the categorical columns into numerical dummy data i.e. convert them into numerical factors which can be used to calculate in the regression analysis.From the summary of the model, we can observe that age and bmi attributes have a higher impact or significance on the overall accuracy of the model. A confusion matrix cannot be applied in regression analysis.Innovation in this experiment includes using regression along with caret package to find out major insights into the field of insurance charges and analysing the behaviour of insurance charges according to the given attributes.