۱.۱

DM.R*

Source on Save | Run | Source

```
1  1-1
2  setwd("/home/nick/Desktop")
3  getwd()
4  data <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
5  summary(data)
6
```

6:1 (Top Level) R Script

Console | Terminal | Jobs

~/Desktop/

```
> setwd("/home/nick/Desktop")
> getwd()
[1] "/home/nick/Desktop"
> data <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
> summary(data)
    T3_resin     Serum_thyroxin     Serum_triiodothyronine  Basal_TSH
 Min.   : 60.0   Length:215         Length:215             Length:215
 1st Qu.:103.0   Class :character   Class :character       Class :character
 Median :110.0   Mode  :character   Mode  :character       Mode  :character
 Mean   :109.8
 3rd Qu.:118.0
 Max.   :180.0
 NA's   :13
 Abs_diff_TSH          Outcome
 Length:215         Min.   :1.00
 Class :character   1st Qu.:1.00
 Mode  :character   Median :1.00
                    Mean   :1.41
                    3rd Qu.:2.00
                    Max.   :3.00
                    NA's   :15
>
```

۱.۲

```
import pandas as pd
aqi_data=pd.read_csv('thyroid.csv')
aqi_data.head(10)
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	107.0	10.1	2.2	0.9	2.7	1.0
1	NaN	9.9	3.1	2.0	5.9	1.0
2	127.0	12.9	2.4	NaN	0.6	1.0
3	109.0	NaN	1.6	1.4	1.5	1.0
4	105.0	7.3	1.5	NaN	-0.1	1.0
5	105.0	6.1	2.1	1.4	7.0	1.0
6	110.0	NaN	1.6	1.6	2.7	1.0
7	114.0	NaN	2.4	1.5	5.7	1.0
8	106.0	?	2.2	1.5	NaN	1.0
9	107.0	13.0	1.1	0.9	3.1	1.0

```
aqi_data.describe(include = 'all')
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
count	202.000000	202	204	201	201.000000	200.000000
unique	NaN	99	49	45	NaN	NaN
top	NaN	8.1	1.8	1.0	NaN	NaN
freq	NaN	8	20	16	NaN	NaN
mean	109.831683	NaN	NaN	NaN	3.999005	1.410000
std	14.068467	NaN	NaN	NaN	7.612509	0.710227
min	60.000000	NaN	NaN	NaN	-4.000000	1.000000
25%	103.000000	NaN	NaN	NaN	0.600000	1.000000
50%	110.000000	NaN	NaN	NaN	2.000000	1.000000
75%	118.000000	NaN	NaN	NaN	4.100000	2.000000
max	180.000000	NaN	NaN	NaN	56.300000	3.000000

۱.۳ در این قسمت خواسته شده که مقادیر یکتای ویژگی های دسته ای را نمایش دهیم. با بررسی مقادیر بنظر می رسد که تنها مقدار outcome دسته ای باشد و بقیه مقادیر همگی از نوع int و یا float باشند ولی ما همگی* را دسته بندی کردیم(برای دسته بندی کردن بر اساس مقادیر ویژه و نمایش فراوانی هر متغیر می توان از دو تابع groupby و یا value_counts استفاده کرد که در تنها در ترتیب نمایش متفاوت اند درواقع value_counts خروجی را بر اساس تعداد فراوانی هر دسته به صورت نزولی نمایش می دهد.)

*** دلیل دسته بندی همه مقادیر این است که ما بررسی کنیم مقادیر هر دسته چگونه هستند و البته در تمرین های بعدی به ما در مدیریت داده های گمشده کمک خواهد کرد.**

```
aqi_data.nunique()
```

```
T3_resin                 56
Serum_thyroxin           99
Serum_triiodothyronine   49
Basal_TSH                45
Abs_diff_TSH             83
Outcome                   3
dtype: int64
```

```
aqi_data.groupby('Outcome').size()
```

```
Outcome
1.0    144
2.0     30
3.0     26
dtype: int64
```

```
aqi_data.groupby('T3_resin').size()
```

```
T3_resin
60.0      1
65.0      1
68.0      1
76.0      1
79.0      1
80.0      1
84.0      2
88.0      3
89.0      4
90.0      1
91.0      1
92.0      1
93.0      1
94.0      2
95.0      1
96.0      1
97.0      4
98.0      5
99.0      3
100.0     4
101.0     5
102.0     6
103.0     7
104.0     3
105.0    10
106.0     9
107.0     5
108.0     6
109.0     7
110.0    11
111.0     4
112.0     7
113.0     6
114.0     8
115.0     6
116.0     6
117.0     5
118.0     6
119.0     7
120.0     9
121.0     4
122.0     2
123.0     2
125.0     2
126.0     3
127.0     2
129.0     2
130.0     1
131.0     1
133.0     1
134.0     3
136.0     1
139.0     2
141.0     2
144.0     1
180.0     1
dtype: int64
```

```
aqi_data.groupby('Serum_thyroxin').size()
```

```
Serum_thyroxin
0.5     1
0.8     1
1.4     1
1.5     1
1.9     3
10.0    3
10.1    4
10.4    7
10.5    2
10.6    4
10.9    2
11.0    1
11.1    7
11.3    2
11.4    1
11.5    2
11.9    3
12.0    1
12.2    2
12.4    2
12.9    3
13.0    4
13.4    1
13.5    1
13.8    1
14.2    1
14.3    1
14.7    1
15.1    1
15.2    1
       ..
5.9     1
6.1     3
6.3     4
6.5     4
6.6     1
6.7     4
6.8     3
7.0     2
7.1     4
7.3     2
7.5     2
7.6     2
7.7     1
7.8     6
8.0     2
8.1     8
8.4     7
8.5     5
8.6     1
8.7     2
8.9     3
9.0     1
9.1     3
9.2     4
9.4     4
9.5     6
9.6     2
9.7     3
9.9     1
?       1
Length: 99, dtype: int64
```

```
aqi_data.groupby('Serum_triiodothyronine').size()
```

```
Serum_triiodothyronine
0.2      1
0.3      2
0.4      1
0.5      1
0.6      2
0.7      5
0.8      1
0.9      1
1.0      5
1.1      9
1.2     12
1.3      6
1.4      9
1.5     18
1.6     14
1.7     11
1.8     20
1.9      8
10.0     2
2.0     11
2.1      8
2.2      5
2.3      7
2.4      7
2.5      3
2.6      1
2.7      5
2.9      2
3.0      1
3.1      3
3.3      2
3.6      2
3.7      1
3.8      1
4.1      1
4.3      1
4.4      1
4.5      1
4.8      1
4.9      1
5.4      1
5.5      2
5.8      1
6.0      1
7.1      1
7.3      1
7.4      1
7.8      1
?        2
dtype: int64
```

```
aqi_data.groupby('Basal_TSH').size()
```

```
Basal_TSH
0.3      2
0.5      5
0.6      5
0.7      8
0.8     14
0.9     13
1.0     16
1.1     14
1.2     14
1.3     16
1.4     10
1.5     12
1.6     14
1.7      5
1.8      6
1.9      7
10.4     1
11.2     1
11.6     1
12.2     2
12.5     1
13.7     1
16.5     1
18.4     1
18.5     1
2.0      5
2.1      4
2.2      3
2.3      1
2.7      1
2.8      1
22.8     1
23.0     1
3.7      1
32.6     1
4.3      1
4.7      1
41.0     1
5.8      1
56.4     1
7.0      1
7.5      1
8.5      2
9.9      1
?        1
dtype: int64
```

```
aqi_data.groupby('Abs_diff_TSH').size()
```

```
Abs_diff_TSH
-4.0     1
-0.7     1
-0.6     2
-0.5     2
-0.3     3
-0.2     8
-0.1     9
 0.0     3
 0.1     7
 0.2     4
 0.3     4
 0.4     1
 0.5     5
 0.6     2
 0.7     1
 0.8     4
 0.9     3
 1.0     4
 1.1     3
 1.2     2
 1.3     3
 1.4     5
 1.5     6
 1.6     3
 1.7     3
 1.8     1
 1.9     7
 2.0     4
 2.1     2
 2.2     5
        ..
 5.1     1
 5.4     1
 5.7     2
 5.9     1
 6.0     1
 6.3     1
 6.4     2
 6.7     2
 6.8     1
 7.0     1
 7.5     1
 7.7     1
 8.2     1
 8.4     1
 8.8     1
 9.5     1
 11.5    1
 12.6    1
 13.7    1
 14.4    2
 19.0    1
 19.6    1
 21.6    1
 22.2    1
 24.0    1
 25.9    1
 38.6    1
 40.8    1
 53.0    1
 56.3    1
Length: 83, dtype: int64
```

۲.۱ ابا بررسی داده ها متوجه می شویم که داده ها گمشده در دیتاست به شیوه های مختلف ذخیره شده اند (برخی به صورت خانه خالی و برخی با NULL و برخی با NA و برخی با ؟) که این اتفاق دلایل مختلفی می تواند داشته باشد مثلاً جمع آوری دیتا از دیتابیس ها با فرمت مختلف و یا این که در دیتابیس مربوطه هنگام ذخیره داده ها کنترلی وجود

نداشته است.

```
2-1
data <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
summary(data)
null2_1 <- function(dataset) {
  dataset[dataset==""]<-NA
  dataset[dataset=="?"]<-NA
  dataset[dataset=="NULL"]<-NA
  miss <- colSums(is.na(dataset))
  return(miss)
}
null2_1(data)

20:1  (Top Level)

Console  Terminal  Jobs
~/Desktop/

Error in matrix(if (is.null(value)) logical() else value, nrow = nr, dimnames = list(rn,  :
  length of 'dimnames' [2] not equal to array extent

> View(data)
> null2_1 <- function(dataset) {
+   dataset[dataset==""]<-NA
+   dataset[dataset=="?"]<-NA
+   dataset[dataset=="NULL"]<-NA
+   miss <- colSums(is.na(dataset))
+   return(miss)
+ }
> null2_1(data)
          T3_resin        Serum_thyroxin Serum_triiodothyronine             Basal_TSH          Abs_diff_TSH               Outcome
                13                    14                    13                    15                    14                    15
```

۲.۲

دو نوع داده از دست رفته وجود دارد:

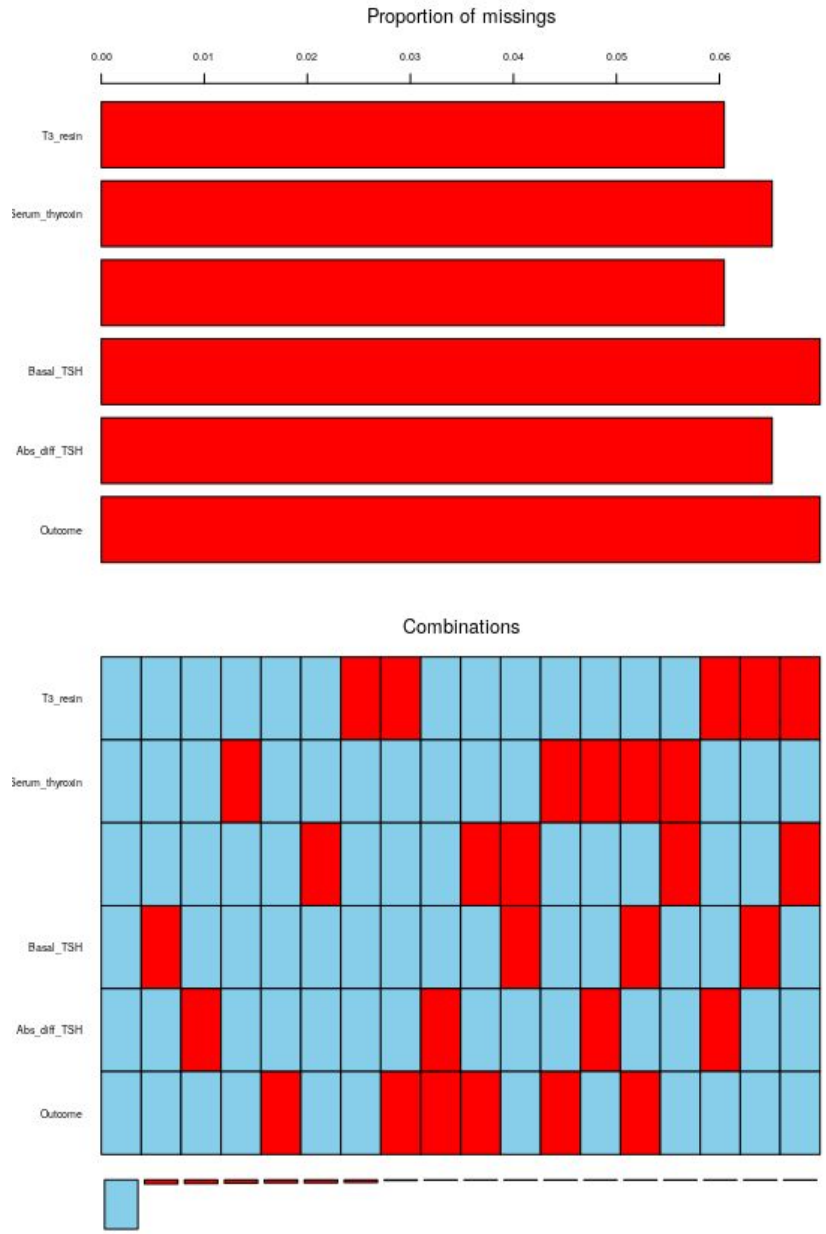
- **MCAR: کاملاً تصادفی از دست می رود. این سناریوی مطلوب در مورد داده های گمشده است.**
- **MNAR: مفقود شدن تصادفی نیست. گم شده در داده های تصادفی نیست یک مسئله جدی تر است و در این مورد ممکن است عاقلانه باشد برای بررسی داده ها فرایند جمع آوری بیشتر و سعی کنید بفهمید چرا اطلاعات از دست رفته است. به عنوان مثال ، اگر بیشتر افراد در یک نظرسنجی به یک سوال خاص پاسخ ندادند ، چرا آنها این کار را کردند؟ آیا این سوال مشخص نبود؟**

استفاده از MICE برای جستجوی الگوی داده های گمشده است.

بسته VIM دارای توابع است که می تواند برای کشف و تحلیل ساختار مقادیر گمشده و نسبت دادن مقدار مناسب با استفاده از روش های گرافیکی؛ مورد استفاده قرار گیرد.

در تحلیل نمودار سمت راست زیر می توان گفت که ۱۴۴ رکورد اول بدون دیتای گمشده است و در ۹ رکورد بعدی اولین داده های گمشده را شاهد هستیم در outcomeو به همین ترتیب تا انتها.
اما نتیجهی نهایی این است که الگویی در گمشده داده ها قابل کشف نیست.
نمودار سمت چپ نشان دهنده درصد داده های گمشده هر متغییر می باشد

```
>
> a=null2_2(data)
> summary(aggr(a))

 Missings per variable:
              Variable Count
              T3_resin    13
        Serum_thyroxin    14
 Serum_triiodothyronine    13
             Basal_TSH    15
          Abs_diff_TSH    14
               Outcome    15

 Missings in combinations of variables:
 Combinations Count    Percent
  0:0:0:0:0:0   144 66.9767442
  0:0:0:0:0:1     9  4.1860465
  0:0:0:0:1:0    11  5.1162791
  0:0:0:0:1:1     1  0.4651163
  0:0:0:1:0:0    12  5.5813953
  0:0:1:0:0:0     9  4.1860465
  0:0:1:0:0:1     1  0.4651163
  0:0:1:1:0:0     1  0.4651163
  0:1:0:0:0:0    10  4.6511628
  0:1:0:0:0:1     1  0.4651163
  0:1:0:0:1:0     1  0.4651163
  0:1:0:1:0:1     1  0.4651163
  0:1:1:0:0:0     1  0.4651163
  1:0:0:0:0:0     8  3.7209302
  1:0:0:0:0:1     2  0.9302326
  1:0:0:0:1:0     1  0.4651163
  1:0:0:1:0:0     1  0.4651163
  1:0:1:0:0:0     1  0.4651163
```


Proportion of missings
0.00
0.01
0.02
0.03
0.04
0.05
0.06
T3_resin
Serum_thyroxin
Basal_TSH
Abs_diff_TSH
Outcome
Combinations
T3_resin
Serum_thyroxin
Basal_TSH
Abs_diff_TSH
Outcome

```r
null2_2 <- function(dataset) {
  dataset[dataset==""]<-NA
  dataset[dataset=="?"]<-NA
  dataset[dataset=="NULL"]<-NA
  return(dataset)
}

a=null2_2(data)
summary(aggr(a))
```

29:17 (Top Level)

Console | Terminal | Jobs

~/Desktop/

```
Missings per variable:
              Variable Count
               T3_resin    13
         Serum_thyroxin    14
Serum_triiodothyronine    13
             Basal_TSH    15
          Abs_diff_TSH    14
               Outcome    15

Missings in combinations of variables:
Combinations Count    Percent
 0:0:0:0:0:0   144 66.9767442
 0:0:0:0:0:1     9  4.1860465
 0:0:0:0:1:0    11  5.1162791
 0:0:0:0:1:1     1  0.4651163
 0:0:0:1:0:0    12  5.5813953
 0:0:1:0:0:0     9  4.1860465
 0:0:1:0:0:1     1  0.4651163
 0:0:1:1:0:0     1  0.4651163
 0:1:0:0:0:0    10  4.6511628
 0:1:0:0:0:1     1  0.4651163
 0:1:0:0:1:0     1  0.4651163
 0:1:0:1:0:1     1  0.4651163
 0:1:1:0:0:0     1  0.4651163
 1:0:0:0:0:0     8  3.7209302
 1:0:0:0:0:1     2  0.9302326
 1:0:0:0:1:0     1  0.4651163
 1:0:0:1:0:0     1  0.4651163
 1:0:1:0:0:0     1  0.4651163
>
```

۲.۳ در قسمت ۲.۱ با مشاهده دیتاست متوجه شدیم که داده های گمشده موجود در دیتاست به صورت های مختلف ذخیره شده اند و مجبور شدیم. هر کدام را به صورت جداگانه شناسایی کنیم اما در پایتون بعد از این که داده ها را در دیتافریم (به کمک کتابخانه pandas) ریختیم مشاهده می کنیم که خانه های خالی و NA و NULL را همگی به صورت داده های گمشده می بیند بنابراین کارمان ساده تر می شود اما اگر با استفاده از dtypes دیتافریم را بررسی کنیم متوجه می شویم برخلاف آنچه ما انتظار داشتیم که همه متغیر ها دارای مقادیر عددی باشند. سه تا از متغیر ها مقادیر دارای نوع object می باشند که نشان می دهد هنوز در این داده ها مقادیری نا معتبر می باشد. با بررسی دقیق تر متوجه می شویم که برخی از خانه ها به جای مقدار عددی مقدار "?" را دارند بنابراین ما تلاش می کنیم که این مقادیر را مقدار NA تبدیل کرده و بعد از آن مشاهده می کنیم که نوع همه ی داده های دیتافریم عددی شده است.

```
aqi_data.empty
```

```
False
```

```
np.sum(aqi_data.applymap(lambda x: x == '?'))
```

```
T3_resin                0
Serum_thyroxin          1
Serum_triiodothyronine  2
Basal_TSH               1
Abs_diff_TSH            0
Outcome                 0
dtype: int64
```

```
aqi_data.groupby('Abs_diff_TSH').size().sum()
```

```
201
```

```
aqi_data.groupby('Serum_triiodothyronine').size().sum()
```

```
204
```

```
aqi_data.groupby('Basal_TSH').size().sum()
```

```
201
```

```
aqi_data.groupby('Outcome').size().sum()
```

```
200
```

```
aqi_data.groupby('T3_resin').size().sum()
```

```
202
```

```
aqi_data.groupby('Serum_thyroxin').size().sum()
```

```
202
```

```
aqi_data.isnull().sum().sum()
```

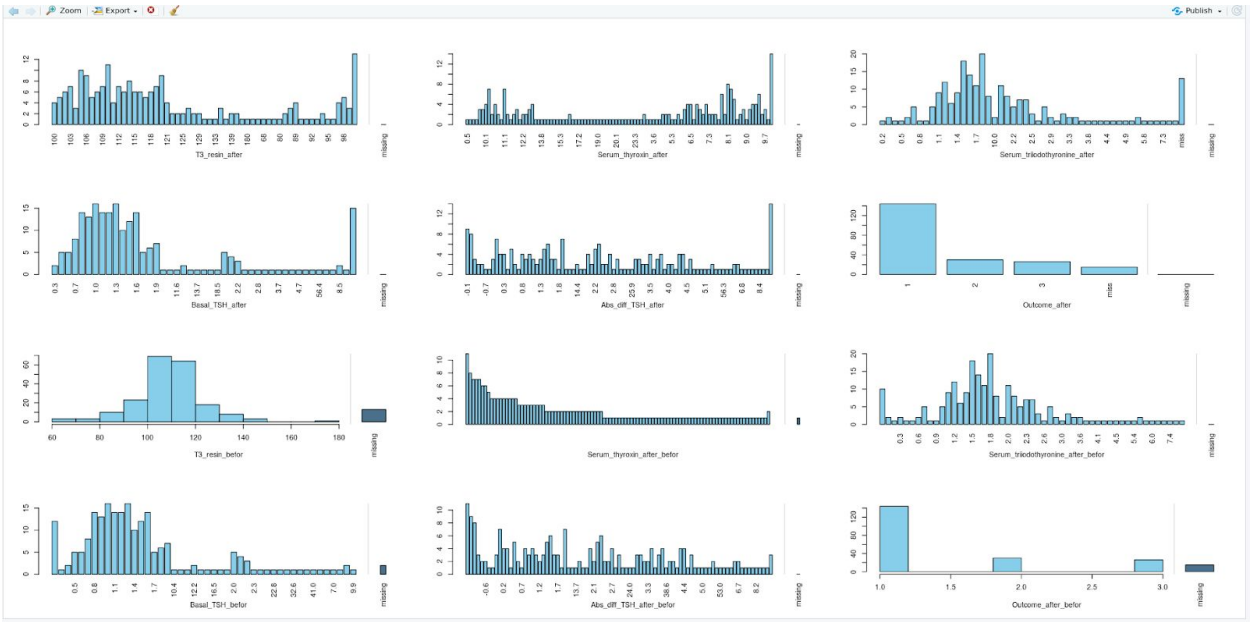
```
80
```

```
1 aqi_data.replace(to_replace ="?", value =np.nan,inplace = True)
```

```
1 aqi_data.isnull().sum().sum()
```

```
84
```

```r
library("VIM")
data <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
null3_1 <- function(dataset) {
  dataset[dataset==""]<-"miss"
  dataset[dataset=="?"]<-"miss"
  dataset[dataset=="NULL"]<-"miss"
  dataset[is.na(dataset)]<-"miss"
  return(dataset)
}

b=null3_1(data)
par(mfrow=c(4,3))
barMiss((b$T3_resin),xlab = "T3_resin_after")
barMiss((b$Serum_thyroxin),xlab ="Serum_thyroxin_after")
barMiss((b$Serum_triiodothyronine),xlab ="Serum_triiodothyronine_after")
barMiss((b$Basal_TSH),xlab ="Basal_TSH_after")
barMiss((b$Abs_diff_TSH),xlab ="Abs_diff_TSH_after")
barMiss((b$Outcome),xlab ="Outcome_after")
barMiss((data$T3_resin),xlab = "T3_resin_befor")
barMiss((data$Serum_thyroxin),xlab ="Serum_thyroxin_after_befor")
barMiss((data$Serum_triiodothyronine),xlab ="Serum_triiodothyronine_after_befor")
barMiss((data$Basal_TSH),xlab ="Basal_TSH_befor")
barMiss((data$Abs_diff_TSH),xlab ="Abs_diff_TSH_after_befor")
barMiss((data$Outcome),xlab ="Outcome_after_befor")
```



۳.۲

```r

data <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
library("VIM")
null2_2 <- function(dataset) {
  dataset[dataset==""]<-NA
  dataset[dataset=="?"]<-NA
  dataset[dataset=="NULL"]<-NA
  return(dataset)
}

b=null2_2(data)
null3_2 <- function(dataset) {
  for( i in seq_along(dataset)){
    dataset[i][is.na(dataset[i])]<-sample(dataset[i][!is.na(dataset[i])],1,replace=FALSE)
  }
  return(dataset)
}

```

106:1 (Top Level)

Console **Terminal** **Jobs**

~/Desktop/

```
> a
    T3_resin Serum_thyroxin Serum_triiodothyronine Basal_TSH Abs_diff_TSH Outcome
1        107           10.1                     2.2       0.9          2.7       1
2         99            9.9                     3.1       2.0          5.9       1
3        127           12.9                     2.4       1.6          0.6       1
4        109            8.9                     1.6       1.4          1.5       1
5        105            7.3                     1.5       1.6         -0.1       1
6        105            6.1                     2.1       1.4          7.0       1
7        110            8.9                     1.6       1.6          2.7       1
8        114            8.9                     2.4       1.5          5.7       1
9        106            8.9                     2.2       1.5          1.0       1
10       107           13.0                     1.1       0.9          3.1       1
11       106            4.2                     1.2       1.6          1.4       1
12       110           11.3                     2.3       0.9          3.3       1
13       116            9.2                     2.7       1.0          4.2       1
14       112            8.1                     1.9       3.7          1.0       1
15       122            9.7                     1.6       0.9          2.2       1
16       109            8.4                     2.1       1.1          1.0       1
17        99            8.4                     1.5       0.8          1.2       1
18       114            6.7                     1.5       1.0          3.5       1
19       119           10.6                     2.1       1.3          1.1       1
20       115            7.1                     1.3       1.3          2.0       1
21       101            7.8                     1.2       1.0          1.7       1
22       103           10.1                     1.5       0.7          0.1       1
23       109           10.4                     1.9       1.6         -0.1       1
24       102            7.6                     1.8       2.0          2.5       1
```

```r
data <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
library("VIM")
null2_2 <- function(dataset) {
  dataset[dataset==""]<-NA
  dataset[dataset=="?"]<-NA
  dataset[dataset=="NULL"]<-NA
  return(dataset)
}

mode_T<-function(colomn_){
  m=names(which.max(table(colomn_)))
  if (is.numeric(colomn_))
    return(as.numeric((m)))
  return (m)
}
b=null2_2(data)
null3_3 <- function(dataset) {
  for( i in seq_along(dataset)){
    dataset[i][is.na(dataset[i])]<-mode_T(dataset[i])
  }
  return(dataset)
}

a=null3_3(b)

a
```

112:1 (Top Level)

Console | **Terminal** | **Jobs**

~/Desktop/

```
+ }
> a=null3_3(b)
> a
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
1	107	10.1	2.2	0.9	2.7	1
2	110	9.9	3.1	2.0	5.9	1
3	127	12.9	2.4	1.0	0.6	1
4	109	8.1	1.6	1.4	1.5	1
5	105	7.3	1.5	1.0	-0.1	1
6	105	6.1	2.1	1.4	7.0	1
7	110	8.1	1.6	1.6	2.7	1
8	114	8.1	2.4	1.5	5.7	1
9	106	8.1	2.2	1.5	-0.1	1
10	107	13.0	1.1	0.9	3.1	1
11	106	4.2	1.2	1.6	1.4	1
12	110	11.3	2.3	0.9	3.3	1
13	116	9.2	2.7	1.0	4.2	1
14	112	8.1	1.9	3.7	-0.1	1
15	122	9.7	1.6	0.9	2.2	1
16	109	8.4	2.1	1.1	-0.1	1
17	110	8.4	1.5	0.8	1.2	1
18	114	6.7	1.5	1.0	3.5	1
19	119	10.6	2.1	1.3	1.1	1

```
data=pd.read_csv('thyroid.csv')
data.fillna("No Data", inplace = True)
```

```
data # changing null value to expression in python dataset
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	107.0	10.1	2.2	0.9	2.7	1.0
1	NaN	9.9	3.1	2.0	5.9	1.0
2	127.0	12.9	2.4	NaN	0.6	1.0
3	109.0	NaN	1.6	1.4	1.5	1.0
4	105.0	7.3	1.5	NaN	-0.1	1.0
5	105.0	6.1	2.1	1.4	7.0	1.0
6	110.0	NaN	1.6	1.6	2.7	1.0
7	114.0	NaN	2.4	1.5	5.7	1.0
8	106.0	?	2.2	1.5	NaN	1.0

```
data  #after changing null value to expression in python dataset
      # Note : "?" still exists
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	107	10.1	2.2	0.9	2.7	1
1	No Data	9.9	3.1	2.0	5.9	1
2	127	12.9	2.4	No Data	0.6	1
3	109	No Data	1.6	1.4	1.5	1
4	105	7.3	1.5	No Data	-0.1	1
5	105	6.1	2.1	1.4	7	1
6	110	No Data	1.6	1.6	2.7	1
7	114	No Data	2.4	1.5	5.7	1
8	106	?	2.2	1.5	No Data	1

```
1 data.replace(to_replace ="?", value ="No Data",inplace = True)
```

```
1 np.sum(data.applymap(lambda x: x == "No Data"))
```

```
T3_resin                 13
Serum_thyroxin           14
Serum_triiodothyronine   13
Basal_TSH                15
Abs_diff_TSH             14
Outcome                  15
dtype: int64
```

```
1 np.sum(data.applymap(lambda x: x == "No Data")).sum()
```

```
84
```

```
1 data
```

0	107	10.1	2.2	0.9	2.7	1
1	No Data	9.9	3.1	2.0	5.9	1
2	127	12.9	2.4	No Data	0.6	1
3	109	No Data	1.6	1.4	1.5	1
4	105	7.3	1.5	No Data	-0.1	1
5	105	6.1	2.1	1.4	7	1
6	110	No Data	1.6	1.6	2.7	1
7	114	No Data	2.4	1.5	5.7	1
8	106	No Data	2.2	1.5	No Data	1
9	107	13.0	1.1	0.9	3.1	1
10	106	4.2	1.2	1.6	1.4	1

```
1 data.dtypes
```

```
T3_resin                 object
Serum_thyroxin           object
Serum_triiodothyronine   object
Basal_TSH                object
Abs_diff_TSH             object
Outcome                  object
dtype: object
```

3.5

```
dat=pd.read_csv('thyroid.csv')
dat
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	107.0	10.1	2.2	0.9	2.7	1.0
1	NaN	9.9	3.1	2.0	5.9	1.0
2	127.0	12.9	2.4	NaN	0.6	1.0
3	109.0	NaN	1.6	1.4	1.5	1.0
4	105.0	7.3	1.5	NaN	-0.1	1.0
5	105.0	6.1	2.1	1.4	7.0	1.0
6	110.0	NaN	1.6	1.6	2.7	1.0
7	114.0	NaN	2.4	1.5	5.7	1.0
8	106.0	?	2.2	1.5	NaN	1.0
9	107.0	13.0	1.1	0.9	3.1	1.0

```
dat.replace('?', np.nan,inplace=True)
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	107.0	10.1	2.2	0.9	2.7	1.0
1	NaN	9.9	3.1	2.0	5.9	1.0
2	127.0	12.9	2.4	NaN	0.6	1.0
3	109.0	NaN	1.6	1.4	1.5	1.0
4	105.0	7.3	1.5	NaN	-0.1	1.0
5	105.0	6.1	2.1	1.4	7.0	1.0
6	110.0	NaN	1.6	1.6	2.7	1.0
7	114.0	NaN	2.4	1.5	5.7	1.0
8	106.0	NaN	2.2	1.5	NaN	1.0
9	107.0	13.0	1.1	0.9	3.1	1.0

```
dat.replace('?', np.nan,inplace=True)
```

```
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
X = dat.values
y=imp.fit_transform(X)
df= pd.DataFrame(y,columns=["T3_resin","Serum_thyroxin","Serum_triiodothyronine","Basal_TSH","Abs_diff_TSH","Outc
df
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	107.000000	10.100000	2.200000	0.9000	2.700000	1.00
1	109.831683	9.900000	3.100000	2.0000	5.900000	1.00
2	127.000000	12.900000	2.400000	2.9815	0.600000	1.00
3	109.000000	9.879104	1.600000	1.4000	1.500000	1.00
4	105.000000	7.300000	1.500000	2.9815	-0.100000	1.00
5	105.000000	6.100000	2.100000	1.4000	7.000000	1.00
6	110.000000	9.879104	1.600000	1.6000	2.700000	1.00
7	114.000000	9.879104	2.400000	1.5000	5.700000	1.00
8	106.000000	9.879104	2.200000	1.5000	3.999005	1.00

۳-۶ برخی از انواع که در کلاس باتوجه به منبع درس ارایه شد:
یک عبارت ثابت جایگذاری شود و یا میانه و مود هر ستون و یا یک عدد رندوم از داده های موجود در ردیف یا روش مهم تر که منظور سوال است یعنی به نوعی تخمین بهتری و نزدیکترین گزینه به مورد نال شده ما با استفاده از داده های دیگر که انواع مختلف به شرح زیر دارد:

Substitution :یک عبارت ثابت

Hot deck imputation: یک مقدار تصادفی با استفاده از مقادیر دیگر موجود در ستون

Cold deck imputation :انتخاب سیستماتیک یک مقدار که مشابه مقادیر دیگر باشد

Regression imputation: پیشبینی مقدار مورد نظر با رگرسیون.

Stochastic regression imputation

Interpolation and extrapolation

4-1

```
1 import numpy as np
2 import pandas as pd
3 dataf=pd.read_csv('housing.csv')
4 dataf.head(10)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY

```
1 dataf.groupby('ocean_proximity').mean()
```

ocean_proximity	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
<1H OCEAN	-118.847766	34.560577	29.279225	2628.343586	546.539185	1520.290499	517.744965	4.230682	240084.285464
INLAND	-119.732990	36.731829	24.271867	2717.742787	533.881619	1391.046252	477.447565	3.208996	124805.392001
ISLAND	-118.354000	33.358000	42.400000	1574.600000	420.400000	668.000000	276.600000	2.744420	380440.000000
NEAR BAY	-122.260694	37.801057	37.730131	2493.589520	514.182819	1230.317467	488.616157	4.172885	259212.311790
NEAR OCEAN	-119.332555	34.738439	29.347254	2583.700903	538.615677	1354.008653	501.244545	4.005785	249433.977427

```
1
```

```
1 import numpy as np
2 import pandas as pd
3 dataf=pd.read_csv('housing.csv')
4 dataf.head(10)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
5	-122.25	37.85	52.0	919.0	213.0	413.0	193.0	4.0368	269700.0	NEAR BAY
6	-122.25	37.84	52.0	2535.0	489.0	1094.0	514.0	3.6591	299200.0	NEAR BAY
7	-122.25	37.84	52.0	3104.0	687.0	1157.0	647.0	3.1200	241400.0	NEAR BAY
8	-122.26	37.84	42.0	2555.0	665.0	1206.0	595.0	2.0804	226700.0	NEAR BAY
9	-122.25	37.84	52.0	3549.0	707.0	1551.0	714.0	3.6912	261100.0	NEAR BAY

```
1 dataf.groupby('ocean_proximity').mean()
```

ocean_proximity	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
<1H OCEAN	-118.847766	34.560577	29.279225	2628.343586	546.539185	1520.290499	517.744965	4.230682	240084.285464
INLAND	-119.732990	36.731829	24.271867	2717.742787	533.881619	1391.046252	477.447565	3.208996	124805.392001
ISLAND	-118.354000	33.358000	42.400000	1574.600000	420.400000	668.000000	276.600000	2.744420	380440.000000
NEAR BAY	-122.260694	37.801057	37.730131	2493.589520	514.182819	1230.317467	488.616157	4.172885	259212.311790
NEAR OCEAN	-119.332555	34.738439	29.347254	2583.700903	538.615677	1354.008653	501.244545	4.005785	249433.977427

5.1

```
dat=pd.read_csv('thyroid.csv')
dat.replace('?', np.nan,inplace=True)
from sklearn.impute import SimpleImputer
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
X = dat.values
y=imp.fit_transform(X)
```

```
from sklearn.preprocessing import Normalizer
scaler = Normalizer().fit(y)
normalized_X = scaler.transform(y)
normalized_X
```

```
array([[0.99497438, 0.09391814, 0.02045742, 0.00836894, 0.02510683,
        0.00929883],
       [0.99394461, 0.0895921 , 0.02805409, 0.01809942, 0.05339327,
        0.00904971],
       [0.99439251, 0.10100522, 0.01879167, 0.02334473, 0.00469792,
        0.00782986],
       ...,
       [0.99702483, 0.04936725, 0.01355179, 0.01161582, 0.04839926,
        0.02903956],
       [0.9897717 , 0.04795801, 0.01122422, 0.02142805, 0.12856828,
        0.0306115 ],
       [0.99624302, 0.05176557, 0.01367392, 0.01269721, 0.06543949,
        0.01377159]])
```

```
df= pd.DataFrame(normalized_X,columns=["T3_resin","Serum_thyroxin","Serum_triiodothyronine","Basal_TSH","Abs_diff
df.head(10)
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	0.994974	0.093918	0.020457	0.008369	0.025107	0.009299
1	0.993945	0.089592	0.028054	0.018099	0.053393	0.009050
2	0.994393	0.101005	0.018792	0.023345	0.004698	0.007830
3	0.995595	0.090235	0.014614	0.012787	0.013701	0.009134
4	0.997045	0.069318	0.014244	0.028311	-0.000950	0.009496
5	0.995783	0.057850	0.019916	0.013277	0.066386	0.009484
6	0.995444	0.089401	0.014479	0.014479	0.024434	0.009049
7	0.994691	0.086199	0.020941	0.013088	0.049735	0.008725
8	0.994629	0.092699	0.020643	0.014075	0.037524	0.009383

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(y)
standardized_X = scaler.transform(y)
standardized_X
```

```
array([[-2.08170445e-01,  4.83423597e-02,  5.83571552e-02,
        -3.43056580e-01, -1.76924186e-01, -6.00039017e-01],
       [-1.04470725e-15,  4.57292592e-03,  6.89864942e-01,
        -1.61763167e-01,  2.58915096e-01, -6.00039017e-01],
       [ 1.26212430e+00,  6.61114433e-01,  1.98692219e-01,
         7.31913166e-17, -4.62943714e-01, -6.00039017e-01],
       ...,
       [-5.02229394e-01, -1.04589349e+00, -5.02983100e-01,
        -2.93612922e-01,  1.36335298e-01,  2.32698058e+00],
       [-9.43317818e-01, -1.13343235e+00, -7.13485695e-01,
        -1.45281948e-01,  1.17145359e+00,  2.32698058e+00],
       [-5.75744131e-01, -1.00212405e+00, -5.02983100e-01,
        -2.77131703e-01,  3.67874916e-01,  0.00000000e+00]])
```

```
df= pd.DataFrame(standardized_X ,columns=["T3_resin","Serum_thyroxin","Serum_triiodothyronine","Basal_TSH","Abs_d
df.head(10)
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	-2.081704e-01	0.048342	0.058357	-3.430566e-01	-1.769242e-01	-0.600039
1	-1.044707e-15	0.004573	0.689865	-1.617632e-01	2.589151e-01	-0.600039
2	1.262124e+00	0.661114	0.198692	7.319132e-17	-4.629437e-01	-0.600039
3	-6.114097e-02	0.000000	-0.362648	-2.606505e-01	-3.403639e-01	-0.600039
4	-3.551999e-01	-0.564430	-0.432816	7.319132e-17	-5.582836e-01	-0.600039
5	-3.551999e-01	-0.827046	-0.011810	-2.606505e-01	4.087348e-01	-0.600039
6	1.237377e-02	0.000000	-0.362648	-2.276880e-01	-1.769242e-01	-0.600039
7	3.064327e-01	0.000000	0.198692	-2.441693e-01	2.316751e-01	-0.600039
8	-2.816852e-01	0.000000	0.058357	-2.441693e-01	-1.209697e-16	-0.600039
9	-2.081704e-01	0.682999	-0.713486	-3.430566e-01	-1.224443e-01	-0.600039

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().fit(y)
MinMaxScaler_X = scaler.transform(y)
MinMaxScaler_X
```

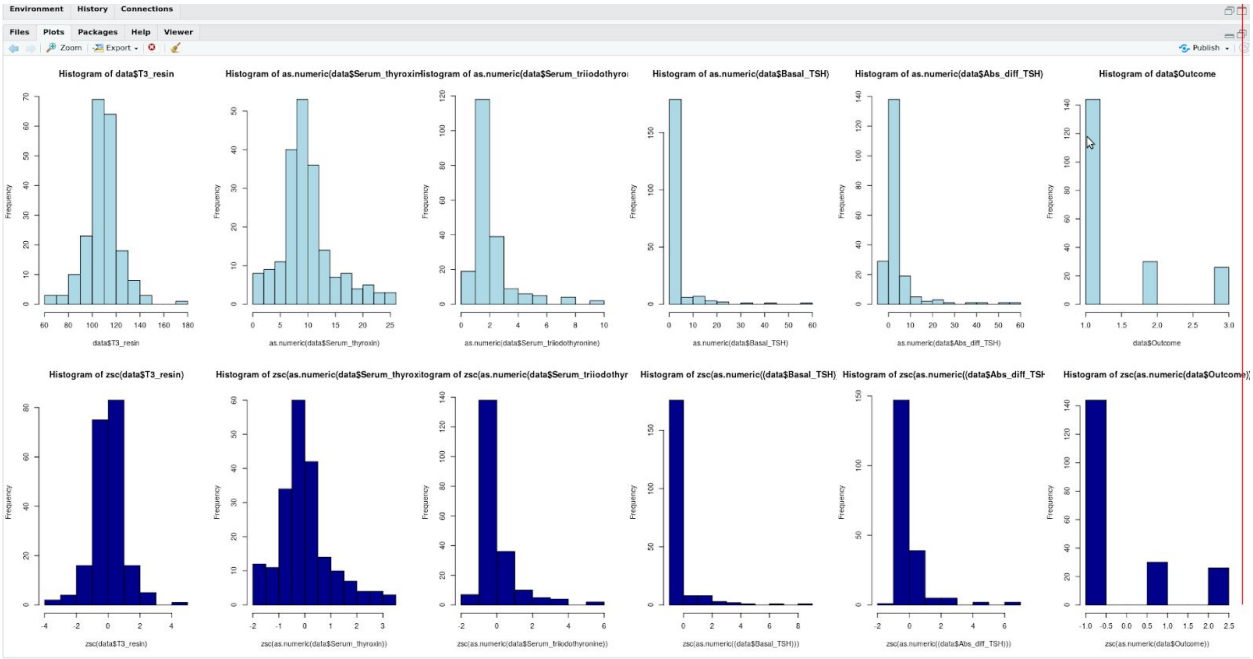
```
array([[0.39166667, 0.38709677, 0.20408163, 0.01069519, 0.11111111,
        0.        ],
       [0.41526403, 0.37903226, 0.29591837, 0.03030303, 0.1641791 ,
        0.        ],
       [0.55833333, 0.5       , 0.2244898 , 0.04779857, 0.07628524,
        0.        ],
       ...,
       [0.35833333, 0.18548387, 0.12244898, 0.01604278, 0.14925373,
        1.        ],
       [0.30833333, 0.16935484, 0.09183673, 0.03208556, 0.27529022,
        1.        ],
       [0.35      , 0.19354839, 0.12244898, 0.01782531, 0.1774461 ,
        0.205     ]])
```

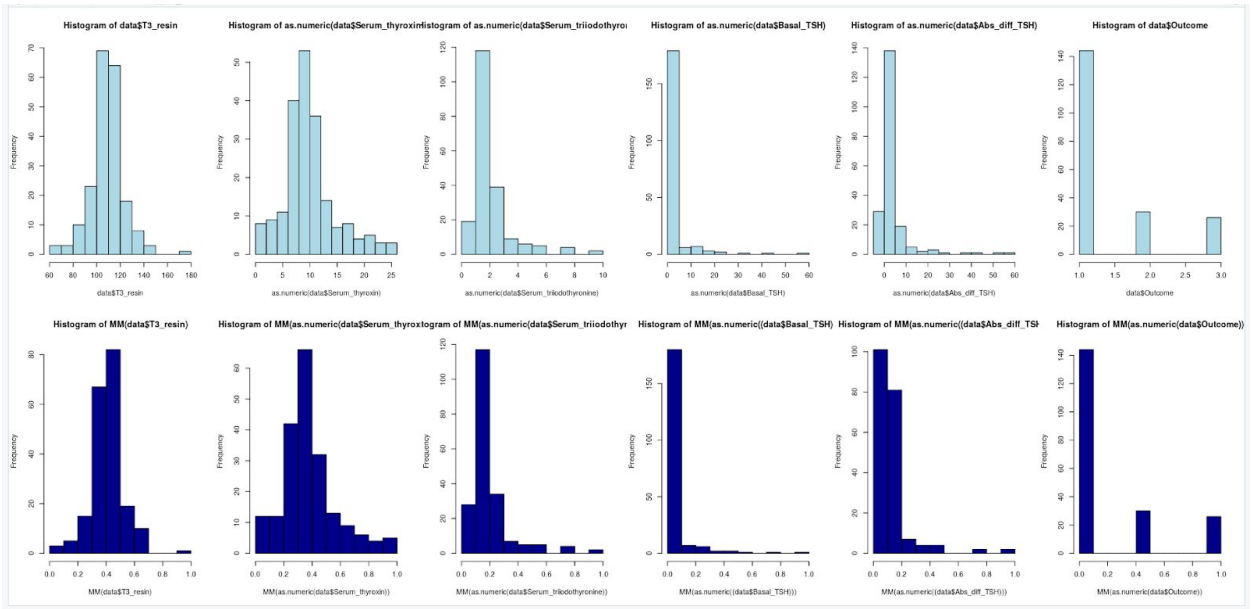
```
df= pd.DataFrame(MinMaxScaler_X ,columns=["T3_resin","Serum_thyroxin","Serum_triiodothyronine","Basal_TSH","Abs_d
df.head(10)
```

	T3_resin	Serum_thyroxin	Serum_triiodothyronine	Basal_TSH	Abs_diff_TSH	Outcome
0	0.391667	0.387097	0.204082	0.010695	0.111111	0.0
1	0.415264	0.379032	0.295918	0.030303	0.164179	0.0
2	0.558333	0.500000	0.224490	0.047799	0.076285	0.0
3	0.408333	0.378190	0.142857	0.019608	0.091211	0.0
4	0.375000	0.274194	0.132653	0.047799	0.064677	0.0
5	0.375000	0.225806	0.193878	0.019608	0.182421	0.0
6	0.416667	0.378190	0.142857	0.023173	0.111111	0.0
7	0.450000	0.378190	0.224490	0.021390	0.160862	0.0
8	0.383333	0.378190	0.204082	0.021390	0.132653	0.0
9	0.391667	0.504032	0.091837	0.010695	0.117745	0.0

5.2

```r
data1 <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
library("VIM")
null2_2 <- function(dataset) {
  dataset[dataset==""]<-NA
  dataset[dataset=="?"]<-NA
  dataset[dataset=="NULL"]<-NA
  return(dataset)
}
data=null2_2(data1)
mean(data$Serum_thyroxin,na.rm =T)
zsc <-function(dataset){
  z<-(dataset-mean(dataset,na.rm =T))/sd(dataset,na.rm=T)
  return(z)
}
MM <- function(dataset){
  mm=((dataset-min(dataset,na.rm = TRUE))/((max(dataset,na.rm = TRUE)-min(dataset,na.rm = TRUE))))
  return(mm)
}

par(mfrow=c(2,6))
hist(data$T3_resin,col = "lightblue")
hist(as.numeric(data$Serum_thyroxin),col = "lightblue")
hist(as.numeric(data$Serum_triiodothyronine),col = "lightblue")
hist(as.numeric(data$Basal_TSH),col = "lightblue")
hist(as.numeric(data$Abs_diff_TSH),col = "lightblue")
hist(data$Outcome,col = "lightblue")
hist(zsc(data$T3_resin),col = "darkblue")
hist(zsc(as.numeric(data$Serum_thyroxin)),col = "darkblue")
hist(zsc(as.numeric(data$Serum_triiodothyronine)),col = "darkblue")
hist(zsc(as.numeric((data$Basal_TSH))),col = "darkblue")
hist(zsc(as.numeric((data$Abs_diff_TSH))),col = "darkblue")
hist(zsc(as.numeric(data$Outcome)),col = "darkblue")

par(mfrow=c(2,6))
hist(data$T3_resin,col = "lightblue")
hist(as.numeric(data$Serum_thyroxin),col = "lightblue")
hist(as.numeric(data$Serum_triiodothyronine),col = "lightblue")
hist(as.numeric(data$Basal_TSH),col = "lightblue")
hist(as.numeric(data$Abs_diff_TSH),col = "lightblue")
hist(data$Outcome,col = "lightblue")
hist(MM(data$T3_resin),col = "darkblue")
hist(MM(as.numeric(data$Serum_thyroxin)),col = "darkblue")
hist(MM(as.numeric(data$Serum_triiodothyronine)),col = "darkblue")
hist(MM(as.numeric((data$Basal_TSH))),col = "darkblue")
hist(MM(as.numeric((data$Abs_diff_TSH))),col = "darkblue")
hist(MM(as.numeric(data$Outcome)),col = "darkblue")
```


Environment
History
Connections
Files
Plots
Packages
Help
Viewer
Zoom
Export
Publish
Histogram of data$T3_resin
Histogram of as.numeric(data$Serum_thyroxin
Histogram of as.numeric(data$Serum_triiodothyro
Histogram of as.numeric(data$Basal_TSH)
Histogram of as.numeric(data$Abs_diff_TSH)
Histogram of data$Outcome
Histogram of zsc(data$T3_resin)
Histogram of zsc(as.numeric(data$Serum_thyrox
Histogram of zsc(as.numeric(data$Serum_triiodothyr
Histogram of zsc(as.numeric((data$Basal_TSH)
Histogram of zsc(as.numeric((data$Abs_diff_TSH
Histogram of zsc(as.numeric(data$Outcome)
Frequency

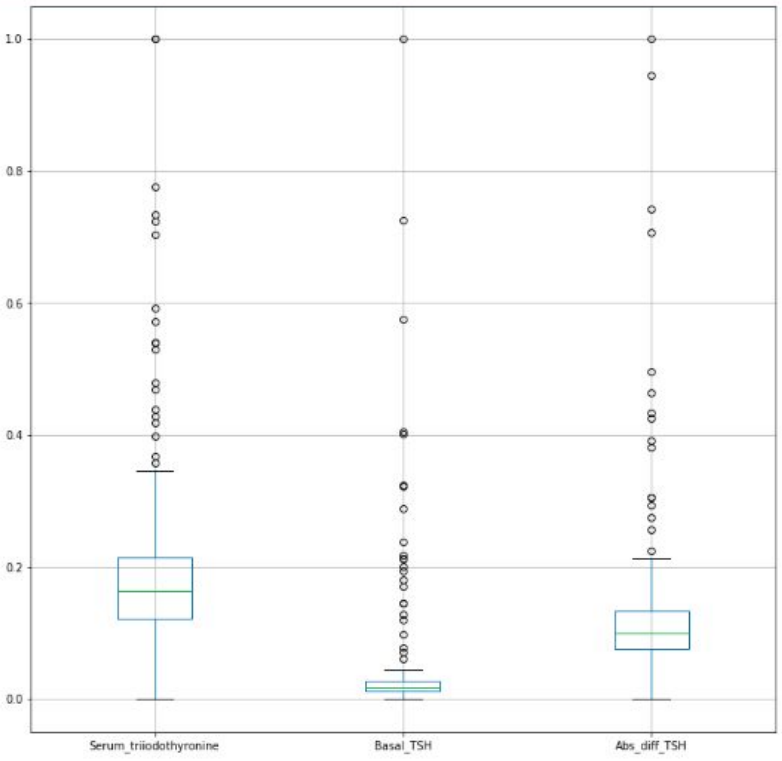

Histogram of data$T3_resin
Histogram of as.numeric(data$Serum_thyroxin
Histogram of as.numeric(data$Serum_triiodothyro
Histogram of as.numeric(data$Basal_TSH)
Histogram of as.numeric(data$Abs_diff_TSH)
Histogram of data$Outcome
Histogram of MM(data$T3_resin)
Histogram of MM(as.numeric(data$Serum_thyrox
Histogram of MM(as.numeric(data$Serum_triiodothyr
Histogram of MM(as.numeric((data$Basal_TSH)
Histogram of MM(as.numeric((data$Abs_diff_TSH
Histogram of MM(as.numeric(data$Outcome))
Frequency

6.1

In [6]:

```
import numpy as np
import pandas as pd

dataset=pd.read_csv('thyroid.csv')
dataset["T3_resin"].values
dataset.replace('?', np.nan,inplace=True)
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().fit(dataset)
MinMaxScaler_X = scaler.transform(dataset)
imp = SimpleImputer(missing_values=np.nan, strategy='mean')
dataset=imp.fit_transform(dataset)
dataset= pd.DataFrame(MinMaxScaler_X ,columns=["T3_resin","Serum_thyroxin","Serum_triiodothyronine","Basal_TSH","
q1=dataset.quantile(0.25)
q3=dataset.quantile(0.75)
lower_bound = q1 - (1.5 * q1)
upper_bound = q3 + (1.5 * q3)
outlier=((dataset<lower_bound) | (dataset>upper_bound))
sum(outlier.values)

boxplot=dataset.boxplot(["Serum_triiodothyronine","Basal_TSH","Abs_diff_TSH"],figsize=(12,12))
```

```
/home/nick/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:323: DataConversionWarning: Data w
ith input dtype float64, object were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```



6.2

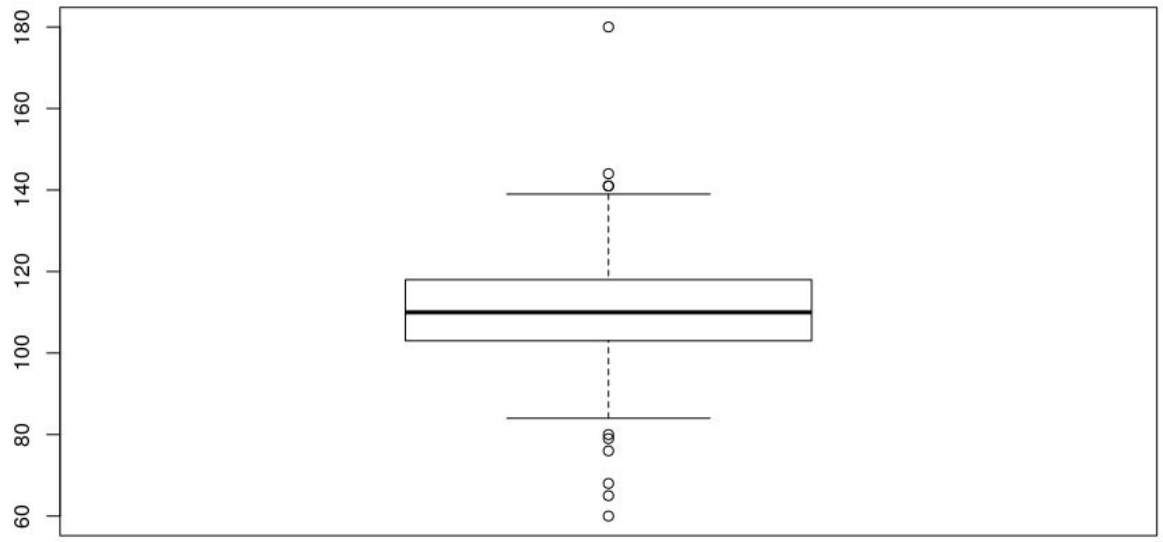
```r
data1 <- read.csv("thyroid.csv", stringsAsFactors = FALSE)
library("VIM")
null2_2 <- function(dataset) {
  dataset[dataset==""]<-NA
  dataset[dataset=="?"]<-NA
  dataset[dataset=="NULL"]<-NA
  return(dataset)
}
data=null2_2(data1)
out<- function(n){
  q1= quantile(data$T3_resin,0.25,na.rm=T)
  q3= quantile(data$T3_resin,0.75,na.rm=T)
  iqr=q3-q1
  upper_=(q3+(1.5*iqr))
  lower_=(q3-(1.5*iqr))
  c(n[n<lower_],n[n>upper_])
}
par(mfrow=c(1,1))
out(data$T3_resin)
boxplot(data$T3_resin,data=data)

```

188:1 (Top Level) R Script

Console **Terminal** **Jobs**

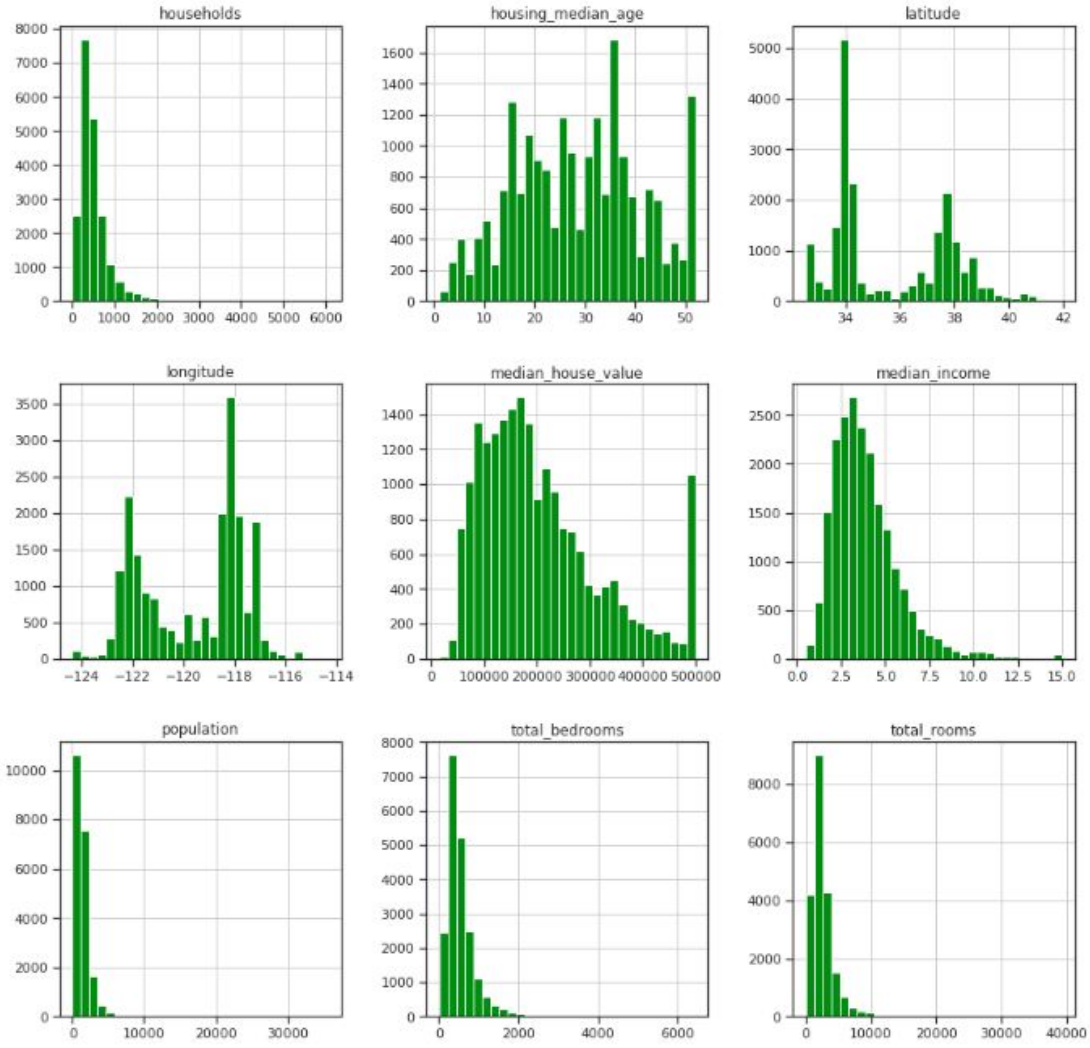
~/Desktop/

```
> out<- function(n){
+   q1= quantile(data$T3_resin,0.25,na.rm=T)
+   q3= quantile(data$T3_resin,0.75,na.rm=T)
+   iqr=q3-q1
+   upper_=(q3+(1.5*iqr))
+   lower_=(q3-(1.5*iqr))
+   c(n[n<lower_],n[n>upper_])
+ }
> out(data$T3_resin)
 [1]  NA  NA  90  NA  93  NA  91  NA  NA  NA  NA  NA  NA  60  NA  94  NA  65  88  NA  95  89  89  88  89  80  89  68  84  84  94  76
  88  79  92  NA  NA  NA  NA  NA  NA
[42]  NA  NA  NA  NA  NA  NA 180  NA 144 141 141
> boxplot(data$T3_resin,data=data)
```



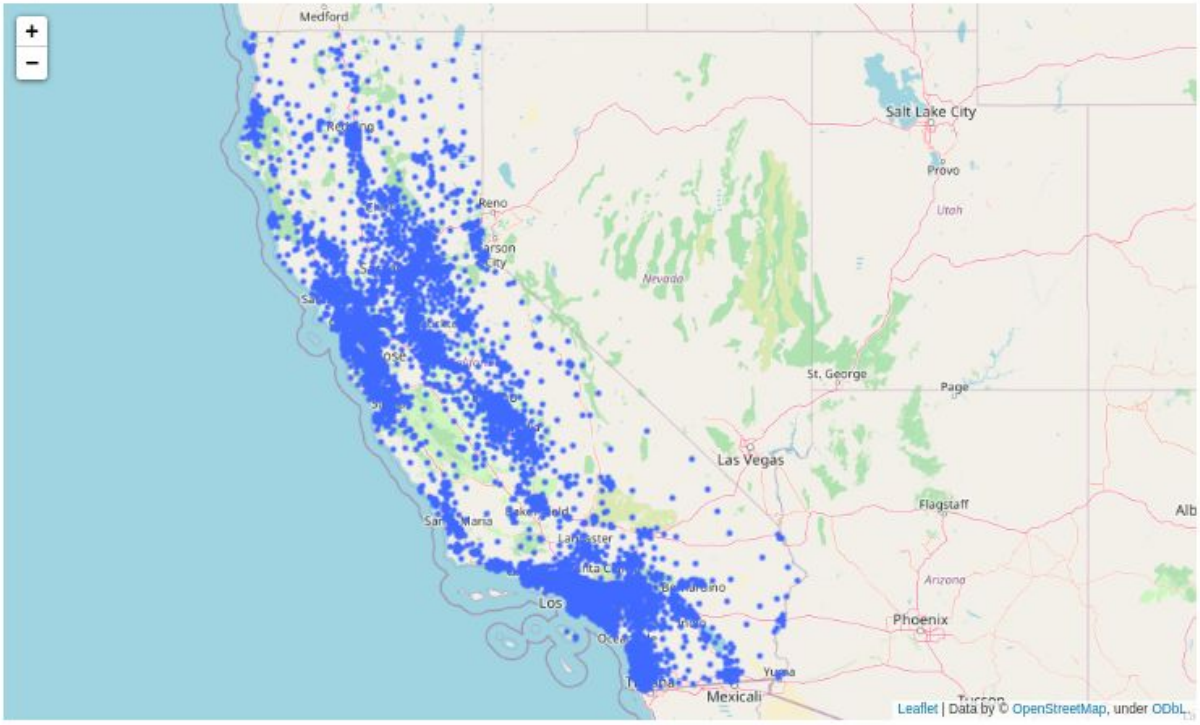
7.1

```
:
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from matplotlib.ticker import StrMethodFormatter
5 dataf=pd.read_csv('housing.csv')
6 fig = plt.figure(figsize = (15,15))
7 ax = fig.gca()
8 dataf.hist(ax=ax,color="green",edgecolor="white", bins=30)
```



7.2

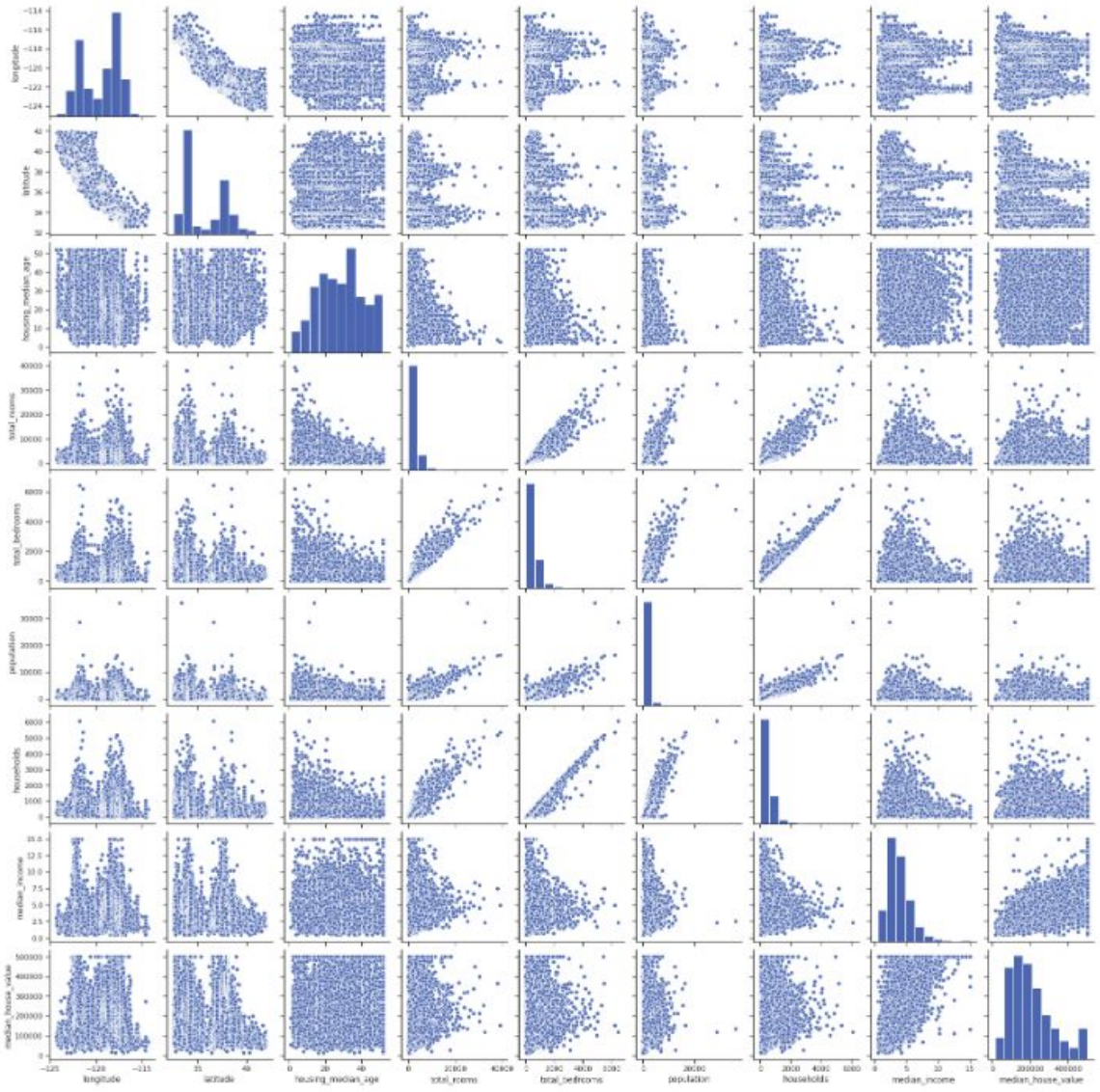
```
: 1  import folium
  2  from folium.plugins import MarkerCluster
  3  import numpy as np
  4  import pandas as pd
  5
  6  df=pd.read_csv('housing.csv')
  7  import folium
  8  from folium.plugins import MarkerCluster
  9  ma=folium.Map(location=[36,-120],zoom_start=5.5)
 10  lat=df['latitude']
 11  lon=df['longitude']
 12  for lat,lon in zip(lat,lon):
 13    folium.CircleMarker(location=[lat,lon],radius=0.5).add_to(ma)
 14
 15  ma
```



8.1

```
: 1 import numpy as np
  2 import pandas as pd
  3 daf=pd.read_csv('housing.csv')
  4 import seaborn as sns
  5 sns.set(style="ticks",color_codes=True)
  6 sns.pairplot(daf)
```

```
: <seaborn.axisgrid.PairGrid at 0x7f8378eda908>
```



8.2

```
: 1 import scipy.stats
  2 scipy.stats.pearsonr(daf["median_house_value"],daf["median_income"])
```

```
: (0.6880752079585479, 0.0)
```

8.3

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter
cdaf=daf.corr(method="pearson")
cdaf
```

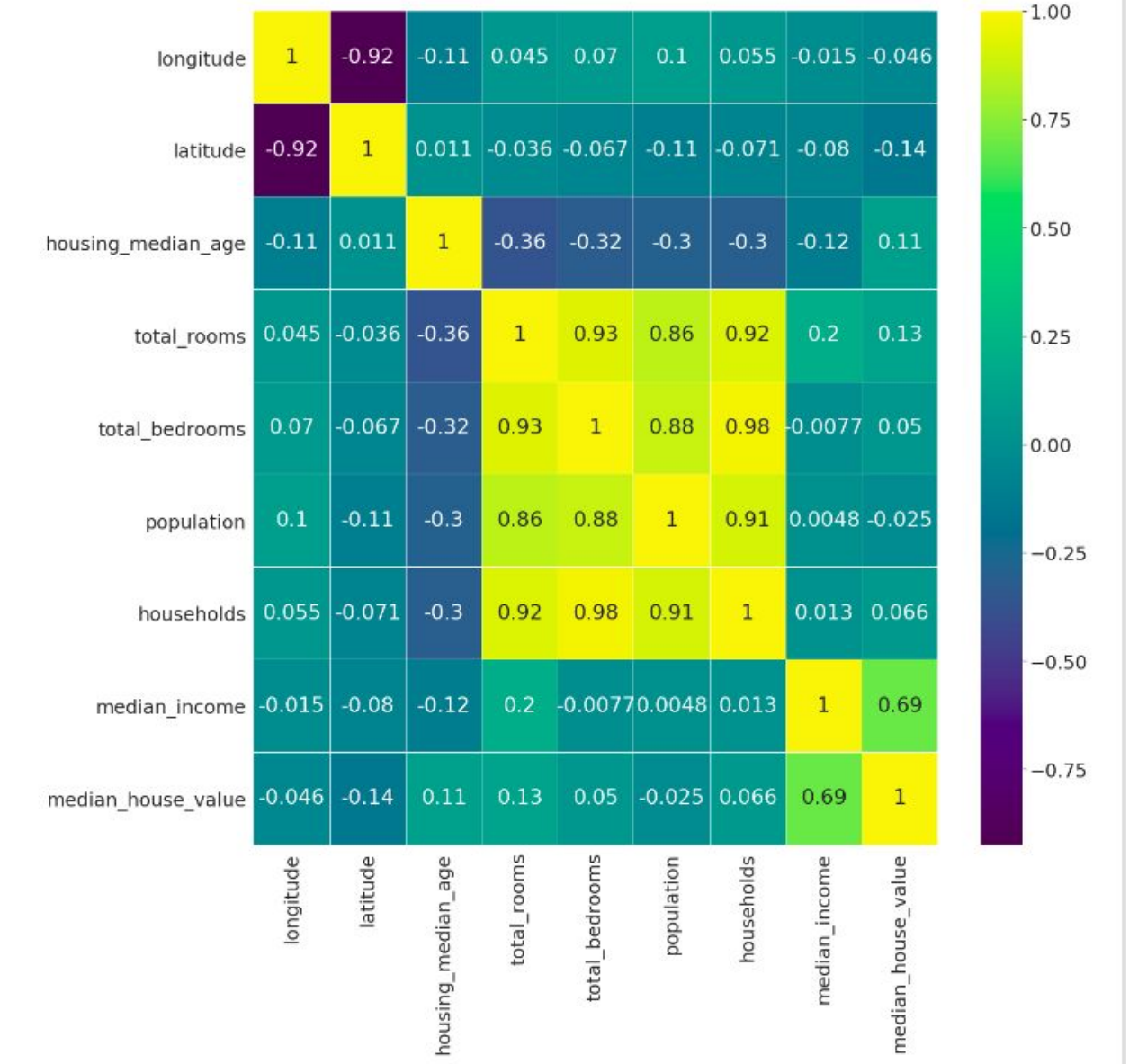
	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608	0.099773	0.055310	-0.015176	-0.045967
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983	-0.108785	-0.071035	-0.079809	-0.144160
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451	-0.296244	-0.302916	-0.119034	0.105623
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380	0.857126	0.918484	0.198050	0.134153
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000	0.877747	0.979728	-0.007723	0.049686
population	0.099773	-0.108785	-0.296244	0.857126	0.877747	1.000000	0.907222	0.004834	-0.024650
households	0.055310	-0.071035	-0.302916	0.918484	0.979728	0.907222	1.000000	0.013033	0.065843
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723	0.004834	0.013033	1.000000	0.688075
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686	-0.024650	0.065843	0.688075	1.000000

8.4

برای بررسی رابطه بین دو متغیر اگر مقدار یک داشته باشد نشان دهنده رابطه خطی هست به ترتیب هر چه دور می شود شکل متفاوتی می گیرد ما به دنبال کشف روابط بین متغیر ها هستیم

```
df_kor = df.corr()
sns.set(font_scale=2.25)
plt.figure(figsize=(20,20))
sns.heatmap(cdaf,cmap="viridis", annot=True, linewidth=0.1)

```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f83797c6fd0>
```



8.5

8.6

8.7

9.1

H0: بین این دو رابطه ای نیست و مستقل اند

H1: این دو با یکدیگر رابطه دارند

9.2

```
1 dfd=pd.read_csv('diabetes.csv')
2 observed=pd.crosstab(dfd["Glucose"],dfd["Outcome"],margins=True)
3 observed
```

Outcome	0	1	All
Glucose			
0	3	2	5
44	1	0	1
56	1	0	1
57	2	0	2
61	1	0	1
62	1	0	1
65	1	0	1
67	1	0	1
68	3	0	3
71	4	0	4
72	1	0	1
73	3	0	3
74	4	0	4
75	2	0	2
76	2	0	2
77	2	0	2
78	3	1	4
79	3	0	3
80	5	1	6
81	6	0	6
82	3	0	3
83	6	0	6

9.3

```
import scipy.stats
scipy.stats.chi2_contingency(observed)
```

```
(269.7332418198132,
 0.5274453807916104,
 272,
 array([[3.25520833e+00, 1.74479167e+00, 5.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [1.30208333e+00, 6.97916667e-01, 2.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [1.95312500e+00, 1.04687500e+00, 3.00000000e+00],
        [2.60416667e+00, 1.39583333e+00, 4.00000000e+00],
        [6.51041667e-01, 3.48958333e-01, 1.00000000e+00],
        [1.95312500e+00, 1.04687500e+00, 3.00000000e+00],
        [2.60416667e+00, 1.39583333e+00, 4.00000000e+00],
        [1.30208333e+00, 6.97916667e-01, 2.00000000e+00],
        [1.30208333e+00, 6.97916667e-01, 2.00000000e+00],
        [1.30208333e+00, 6.97916667e-01, 2.00000000e+00],
        [2.60416667e+00, 1.39583333e+00, 4.00000000e+00],
        [1.95312500e+00, 1.04687500e+00, 3.00000000e+00],
        [3.90625000e+00, 2.09375000e+00, 6.00000000e+00],
        [3.90625000e+00, 2.09375000e+00, 6.00000000e+00],
        [1.95312500e+00, 1.04687500e+00, 3.00000000e+00],
        [3.90625000e+00, 2.09375000e+00, 6.00000000e+00],
        [6.51041667e+00, 3.48958333e+00, 1.00000000e+01],
        [4.55729167e+00, 2.44270833e+00, 7.00000000e+00],
        [1.95312500e+00, 1.04687500e+00, 3.00000000e+00],
        [4.55729167e+00, 2.44270833e+00, 7.00000000e+00],
        [5.85937500e+00, 3.14062500e+00, 9.00000000e+00],
        [3.90625000e+00, 2.09375000e+00, 6.00000000e+00],
        [7.16145833e+00, 3.83854167e+00, 1.10000000e+01],
        [5.85937500e+00, 3.14062500e+00, 9.00000000e+00],
        [5.85937500e+00, 3.14062500e+00, 9.00000000e+00],
        [4.55729167e+00, 2.44270833e+00, 7.00000000e+00],
        [4.55729167e+00, 2.44270833e+00, 7.00000000e+00],
        [8.46354167e+00, 4.53645833e+00, 1.30000000e+01],
```

9.4

```
data1 <- read.csv("diabetes.csv", stringsAsFactors = FALSE)
observed<-table(data1$Glucose,data1$Outcome)
observed
chisq.test(observed)
```

```
        Pearson's Chi-squared test

data:  observed
X-squared = 269.73, df = 135, p-value = 5.105e-11
```

9.5

10.1

```
da=pd.read_csv('diabetes.csv')
da.drop( da[ da['Glucose'] == 0 ].index , inplace=True)
da.drop( da[ da['BloodPressure'] == 0 ].index , inplace=True)
```

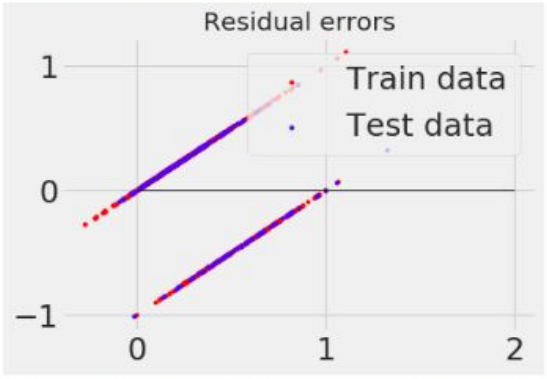
```
da.nunique()
```

```
Pregnancies                 17
Glucose                    135
BloodPressure               46
SkinThickness               51
Insulin                    186
BMI                        246
DiabetesPedigreeFunction   502
Age                         51
Outcome                      2
dtype: int64
```

10.2

```
from statsmodels.tools.eval_measures import mse, rmse
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
import pandas as pd
df = pd.read_csv("diabetes.csv")
df.head()
df.drop( df[ df['Glucose'] == 0 ].index , inplace=True)
df.drop( df[ df['BloodPressure'] == 0 ].index , inplace=True)

X = df[["Pregnancies", "Glucose","Insulin","BMI","DiabetesPedigreeFunction"]]
y = df["Outcome"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
print('Coefficients: \n', reg.coef_)
print('Variance score: {}'.format(reg.score(X_test, y_test)))
plt.style.use('fivethirtyeight')
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,color = "red", s = 10, label = 'Train data')
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,color = "blue", s = 10, label = 'Test data')
plt.hlines(y = 0, xmin = 0, xmax = 2, linewidth = 1)
plt.legend(loc = 'upper right')
plt.title("Residual errors")
plt.show()
y_preds = reg.predict(X_test)
print("Mean Absolute Error (MAE)         : {}".format(mean_absolute_error(y_test, y_preds)))
print("Mean Squared Error (MSE) : {}".format(mse(y_test, y_preds)))
print("Root Mean Squared Error (RMSE) : {}".format(rmse(y_test, y_preds)))
print("Root Mean Squared Error (RMSE) : {}".format(rmse(y_test, y_preds)))
print("Mean Absolute Perc. Error (MAPE) : {}".format(np.mean(np.abs((y_test - y_preds) / y_test)) * 100))

```

```
Coefficients:
 [ 2.32444087e-02  5.98264109e-03 -1.21594186e-04  1.35480351e-02
  1.79299264e-01]
Variance score: 0.3487983549571473
```

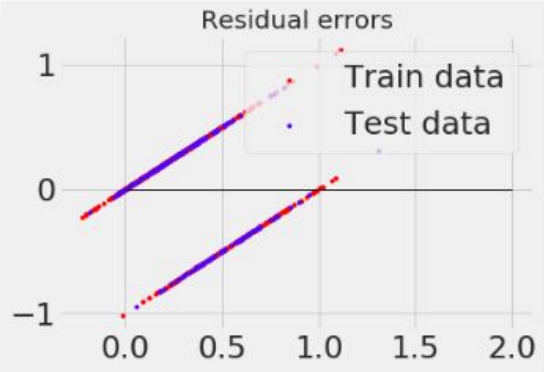



```
Mean Absolute Error (MAE)         : 0.319842252561717
Mean Squared Error (MSE) : 0.1484686276827564
Root Mean Squared Error (RMSE) : 0.3853162696834334
Root Mean Squared Error (RMSE) : 0.3853162696834334
Mean Absolute Perc. Error (MAPE) : inf
```

10.3

```
da2 = da.values
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler().fit(da2)
MinMaxScaler_X = scaler.transform(da2)
df = pd.read_csv("diabetes.csv")
df.head()
df.drop( df[ df['Glucose'] == 0 ].index , inplace=True)
df.drop( df[ df['BloodPressure'] == 0 ].index , inplace=True)

df= pd.DataFrame(da2,columns=["Pregnancies", "Glucose", "BloodPressure", 'SkinThickness',"Insulin","BMI","Diabetes
X = df[["Pregnancies", "Glucose","Insulin","BMI","DiabetesPedigreeFunction"]]
y = df["Outcome"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
print('Coefficients: \n', reg.coef_)
print('Variance score: {}'.format(reg.score(X_test, y_test)))
plt.style.use('fivethirtyeight')
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train,color = "red", s = 10, label = 'Train data')
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,color = "blue", s = 10, label = 'Test data')
plt.hlines(y = 0, xmin = 0, xmax = 2, linewidth = 1)
plt.legend(loc = 'upper right')
plt.title("Residual errors")
plt.show()
y_preds = reg.predict(X_test)
print("Mean Absolute Error (MAE)         : {}".format(mean_absolute_error(y_test, y_preds)))
print("Mean Squared Error (MSE) : {}".format(mse(y_test, y_preds)))
print("Root Mean Squared Error (RMSE) : {}".format(rmse(y_test, y_preds)))
print("Root Mean Squared Error (RMSE) : {}".format(rmse(y_test, y_preds)))
print("Mean Absolute Perc. Error (MAPE) : {}".format(np.mean(np.abs((y_test - y_preds) / y_test)) * 100))
```

```
Coefficients:
 [ 2.53040759e-02  6.22985529e-03 -1.97473901e-04  1.05071298e-02
  2.08515311e-01]
Variance score: 0.29086039853362555
```

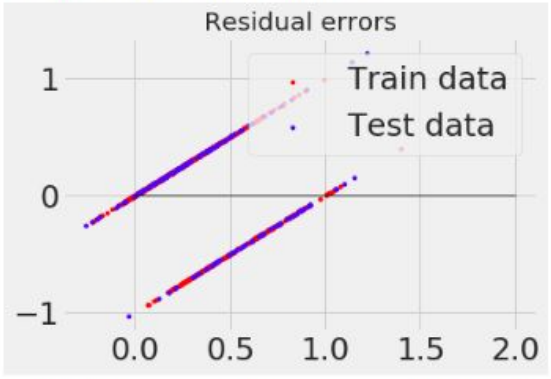



```
Mean Absolute Error (MAE)         : 0.34578735153923384
Mean Squared Error (MSE) : 0.1628126636019737
Root Mean Squared Error (RMSE) : 0.4035005125176097
Root Mean Squared Error (RMSE) : 0.4035005125176097
Mean Absolute Perc. Error (MAPE) : inf
```

10.4

```
da3 = da.values
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(da3)
standardized_X = scaler.transform(da3)
df = pd.read_csv("diabetes.csv")
df.head()
df.drop( df[ df['Glucose'] == 0 ].index , inplace=True)
df.drop( df[ df['BloodPressure'] == 0 ].index , inplace=True)

df= pd.DataFrame(da3,columns=["Pregnancies", "Glucose", "BloodPressure", 'SkinThickness',"Insulin","BMI","Diabetes
X = df[["Pregnancies", "Glucose","Insulin","BMI","DiabetesPedigreeFunction"]]
y = df["Outcome"]
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
reg = linear_model.LinearRegression()
reg.fit(X_train, y_train)
print('Coefficients: \n', reg.coef_)
print('Variance score: {}'.format(reg.score(X_test, y_test)))
plt.style.use('fivethirtyeight')
plt.scatter(reg.predict(X_train), reg.predict(X_train) - y_train, color = "red", s = 10, label = 'Train data')
plt.scatter(reg.predict(X_test), reg.predict(X_test) - y_test,color = "blue", s = 10, label = 'Test data')
plt.hlines(y = 0, xmin = 0, xmax = 2, linewidth = 1)
plt.legend(loc = 'upper right')
plt.title("Residual errors")
plt.show()
y_preds = reg.predict(X_test)
print("Mean Absolute Error (MAE)         : {}".format(mean_absolute_error(y_test, y_preds)))
print("Mean Squared Error (MSE) : {}".format(mse(y_test, y_preds)))
print("Root Mean Squared Error (RMSE) : {}".format(rmse(y_test, y_preds)))
print("Root Mean Squared Error (RMSE) : {}".format(rmse(y_test, y_preds)))
print("Mean Absolute Perc. Error (MAPE) : {}".format(np.mean(np.abs((y_test - y_preds) / y_test)) * 100))

```

```
Coefficients:
 [ 2.11768433e-02  6.92173929e-03 -1.58977902e-04  1.08755877e-02
  2.15112892e-01]
Variance score: 0.3051742288887802
```




```
Mean Absolute Error (MAE)         : 0.3216619138440027
Mean Squared Error (MSE) : 0.15726086239647433
Root Mean Squared Error (RMSE) : 0.3965612971489708
Root Mean Squared Error (RMSE) : 0.3965612971489708
Mean Absolute Perc. Error (MAPE) : inf
```

10.5

11.1

11.2

```
from sklearn.model_selection import train_test_split
Y=df["Outcome"]

X=df[[ "Pregnancies", "Glucose", "BloodPressure", 'SkinThickness',"Insulin","BMI","DiabetesPedigreeFunction","Age'
Xtr, Xte, ytr,yte = train_test_split(X, Y,test_size = 0.2)
Xtr.shape
```

```
(582, 8)
```

11.3

11.4

12.1

12-1

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
import pandas as pd
df = pd.read_csv("corona.csv")
```

12.2

```
df.describe(include = 'all')
```

	age	sex	city	province	country	latitude	longitude	lives_in_Wuhan	outcome
count	651	651	601	526	649	651.000000	651.000000	162	651
unique	101	2	141	77	26	NaN	NaN	2	25
top	30	male	National Centre for Infectious Diseases	Metro Manila	Philippines	NaN	NaN	yes	discharge
freq	18	390	44	107	137	NaN	NaN	89	153
mean	NaN	NaN	NaN	NaN	NaN	17.122780	79.475471	NaN	NaN
std	NaN	NaN	NaN	NaN	NaN	18.458948	67.662144	NaN	NaN
min	NaN	NaN	NaN	NaN	NaN	-34.928900	-123.100000	NaN	NaN
25%	NaN	NaN	NaN	NaN	NaN	5.550000	38.740000	NaN	NaN
50%	NaN	NaN	NaN	NaN	NaN	14.600000	105.678812	NaN	NaN
75%	NaN	NaN	NaN	NaN	NaN	33.649300	120.977200	NaN	NaN
max	NaN	NaN	NaN	NaN	NaN	49.250000	153.400000	NaN	NaN

12.3

```
df["outcome"].value_counts()
```

```
discharge                                                          153
died                                                               141
discharged                                                          74
stable                                                              66
recovered                                                           57
stable condition                                                    46
death                                                               40
Stable                                                              16
Alive                                                               13
Under treatment                                                     12
Dead                                                                 6
Recovered                                                            4
dead                                                                 3
released from quarantine                                             2
Died                                                                 2
severe                                                               2
not hospitalized                                                     2
treated in an intensive care unit (14.02.2020)                       2
recovering at home 03.03.2020                                        2
Symptoms only improved with cough. Currently hospitalized for follow-up.    2
Discharged                                                           2
critical condition                                                   1
Death                                                                1
Receiving Treatment                                                  1
Deceased                                                             1
Name: outcome, dtype: int64
```

12.4

```
df.isnull().sum()
```

```
age                0
sex                0
city              50
province         125
country            2
latitude           0
longitude          0
lives_in_Wuhan   489
outcome            0
dtype: int64
```

```
Recovered=["discharge","recovered","discharged","not hospitalized",
           "treated in an intensive care unit (14.02.2020)",
           "recovering at home 03.03.2020","released from quarantine","Discharged","Alive"]


Under_treatment=["stable condition","stable",
                 "Symptoms only improved with cough. Currently hospitalized for follow-up.",
                "severe","critical condition","Stable","Receiving Treatment"]


Deceased=["died","death","dead","Death","Died","Deceased","Dead"]

for i in Recovered:
    df.loc[df["outcome"]==i,"outcome"]="Recovered"


for i in Under_treatment:
    df.loc[df["outcome"]==i,"outcome"]="Under treatment"


for i in Deceased:
    df.loc[df["outcome"]==i,"outcome"]="Deceased"

df["outcome"].value_counts()
```

```
Recovered          311
Deceased           194
Under treatment    146
Name: outcome, dtype: int64
```

12.5

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model, metrics
import pandas as pd
df = pd.read_csv("corona.csv")
df.loc[df["age"]=="80-","age"]=80
df.loc[df["age"]=="90-99","age"]=95
df.loc[df["age"]=="80-89","age"]=85
df.loc[df["age"]=="70-79","age"]=75
df.loc[df["age"]=="60-69","age"]=65
df.loc[df["age"]=="50-59","age"]=55
df.loc[df["age"]=="40-49","age"]=45
df.loc[df["age"]=="20-29","age"]=25
df.drop([215],inplace=True)            #df[df.age == '15-88']
df.drop([174,337],inplace=True)        #df[df.age == "0.25"]
df.drop([286],inplace=True)             #df[df.age == "0.75"]
df["age"].value_counts()
cv={"age": float}
df=df.astype(cv)
```

```
df.loc[df["latitude"]==43.93907,"city"]="San Marino"     #df[df.latitude == 43.93907]
df.loc[df["latitude"]==1.35346,"city"]="Singapore"      #df[df.latitude == 1.35346]
df.loc[df['city'].isnull(),'city'] = df['province']
df.loc[df['province'].isnull(),'province'] = df['country']
df.country.fillna("Taiwan", inplace=True)
df.drop([45,534],inplace=True)                      #The coordinates are in the ocean
```

```
df.loc[df['city']=='Wuhan City',"lives_in_Wuhan"] = "yes"
df.loc[df['country']!='chin',"lives_in_Wuhan"] = "no"
null_columns=df.columns[df.isnull().any()]
df.isnull().sum().sum
```

```
<bound method Series.sum of age              0
sex              0
city             0
province         0
country          0
latitude         0
longitude        0
lives_in_Wuhan   0
outcome          0
dtype: int64>
```

12.6

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

df["country"]= le.fit_transform(df['country'])
enc = OneHotEncoder(handle_unknown='ignore')
```

```
from sklearn import preprocessing
le = preprocessing.LabelEncoder()
df1=df.apply(le.fit_transform)
df1
```

	age	sex	city	province	country	latitude	longitude	lives_in_Wuhan	outcome
0	71	1	145	76	11	143	31	0	13
1	54	0	118	66	19	23	56	0	15
2	21	1	118	66	19	23	56	0	15
3	49	0	118	66	19	23	56	0	15
4	72	0	86	66	19	24	67	0	15
5	19	1	118	66	19	23	56	0	15
6	18	1	38	66	19	28	69	0	15
7	33	1	150	66	19	31	54	0	15
8	36	1	75	66	19	20	65	0	15
9	22	1	87	66	19	11	55	0	15
10	64	0	94	66	19	16	63	0	15
11	61	0	31	44	11	142	28	0	13
12	28	1	94	66	19	16	63	0	15
13	29	1	27	54	20	126	139	0	15
14	25	1	94	66	19	16	63	0	15
15	23	0	55	66	19	27	70	0	15
16	34	1	94	66	19	16	63	0	15
17	51	0	94	66	19	16	63	0	15
18	31	1	96	66	19	18	52	0	15
19	81	1	112	63	18	137	32	0	13
20	15	0	59	37	20	104	134	0	15
21	48	0	19	60	1	3	149	0	15
22	7	1	63	66	19	14	62	0	15
23	61	0	126	66	19	15	61	0	15
24	47	0	19	60	1	3	149	0	15
25	50	0	16	60	1	4	150	0	15
26	17	1	94	66	19	16	63	0	15
27	29	1	94	66	19	16	63	0	15
28	37	0	152	17	4	99	99	0	16
29	85	0	77	52	1	1	148	0	13
...	...	...	...	...	...	...	...	...	...

12.7

```
from sklearn.model_selection import train_test_split
Y=df["outcome"]

X=df[[ "age","sex","city","province","country","latitude","longitude","lives_in_Wuhan"]]
X_train, X_test, y_train, y_test = train_test_split(X, Y)  #test_size by default = 0.25
```

```
from sklearn.model_selection import train_test_split
Y=df1["outcome"]

X=df1[[ "age","sex","country"]]
X_train, X_test, y_train, y_test = train_test_split(X, Y)  #test_size by default = 0.25
```

12.8

```
from sklearn.model_selection import train_test_split
Y=df["outcome"]
X=df[[ "age","sex","city","province","country","latitude","longitude","lives_in_Wuhan"]]
X_train, X_test, y_train, y_test = train_test_split(X, Y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape    #test_size by default = 0.25. , we can change it.
```

```
((483, 8), (162, 8), (483,), (162,))
```

```
from sklearn.model_selection import train_test_split
Y=df1["outcome"]
X=df1[[ "age","sex","country"]]
X_train, X_test, y_train, y_test = train_test_split(X, Y)
X_train.shape, X_test.shape, y_train.shape, y_test.shape    #test_size by default = 0.25. , we can change it.
```

```
((483, 3), (162, 3), (483,), (162,))
```

12.9

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
x=classifier.fit(X_train, y_train)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 0  0  0  0  0  0  0  0  0  0  1  0  2  1  0  0  0  0  0  2  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  2  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  2  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  1  4  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  2  2  0  2  5 13  2  0  1  0  0  4  1  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  4  9 10  5  0  3  1  0  7  2  0]
 [ 0  0  0  0  0  0  0  0  1  1  0  0  5  6  2  1  0  0  0  4  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  1  0  0  2  2  6  2  0  1  1  0  1  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 1  0  0  0  0  0  0  0  0  0  1  4  3  0  0  1  0  0  2  2  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  3  1  0  0  0  0  1  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         6
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         1
           7       0.00      0.00      0.00         2
           8       0.00      0.00      0.00         3
          10       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         1
          13       0.00      0.00      0.00         6
          14       0.15      0.15      0.15        33
          15       0.19      0.24      0.21        42
          16       0.15      0.10      0.12        20
          17       0.00      0.00      0.00         0
          18       0.17      0.06      0.08        18
          19       0.00      0.00      0.00         1
          20       0.00      0.00      0.00         1
          22       0.11      0.14      0.12        14
          23       0.09      0.17      0.12         6
          24       0.00      0.00      0.00         1

   micro avg       0.13      0.13      0.13       162
   macro avg       0.04      0.04      0.04       162
weighted avg       0.13      0.13      0.12       162
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
x=classifier.fit(X_train, y_train)
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 0  0  0  0  0  0  0  0  1  0  0  0  0  1  0  0  1  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  2  1  0  0  0  0  1  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  2  0  0  0  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  1  3  2  3  0  0  0  0  4  0  0]
 [ 2  0  0  0  1  0  0  0  3  6 12  5  0  3  0  0  3  1  0]
 [ 1  0  0  0  1  0  0  0  2 10  6  4  0  4  0  1  2  3  0]
 [ 0  1  0  0  0  0  0  0  2  5  6  2  0  2  0  0  2  3  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  4  1  0  1  0  0  0  3  0]
 [ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  1  3  8  1  0  1  0  0  1  1  0]
 [ 0  0  0  0  0  0  0  0  0  5  2  0  0  1  0  0  2  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         3
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           8       0.00      0.00      0.00         4
           9       0.00      0.00      0.00         2
          10       0.00      0.00      0.00         3
          12       0.00      0.00      0.00         1
          13       0.08      0.07      0.08        14
          14       0.17      0.17      0.17        36
          15       0.14      0.18      0.15        34
          16       0.12      0.09      0.10        23
          17       0.00      0.00      0.00         0
          18       0.06      0.10      0.08        10
          19       0.00      0.00      0.00         1
          21       0.00      0.00      0.00         0
          22       0.06      0.06      0.06        17
          23       0.00      0.00      0.00        10
          24       0.00      0.00      0.00         1

    micro avg       0.10      0.10      0.10       162
    macro avg       0.03      0.03      0.03       162
 weighted avg       0.10      0.10      0.10       162
```

12.10

```
y_pred = classifier.predict(X_test)
y_pred
```

```
ray([22, 13, 18, 15, 13, 15,  1, 22, 15, 13, 14, 14, 14, 16, 16, 14, 16,
     13, 22, 15, 21, 15, 16, 13, 15, 23, 22, 14, 15, 14, 18, 22, 14, 15,
     14, 23, 23, 14, 22, 16, 15, 18, 15, 13, 15, 15, 15, 14, 14, 14, 15,
     23, 14, 14, 15, 15, 16, 15, 18,  0, 15, 15, 14, 14, 23, 16, 13, 15,
      0, 14, 14, 14, 23, 18, 18, 16, 14, 23, 18, 15, 15, 15, 15, 13, 22,
     23, 23, 15, 16, 15, 15, 16, 22, 13,  8, 16, 14, 18, 22, 15, 14, 16,
     14, 13, 18, 14, 22, 16, 22, 15, 16, 15, 22, 14, 14, 14, 23, 16, 18,
     14,  0,  0, 13, 14,  0, 16, 15, 15, 18, 14, 22, 15, 23, 14, 14, 15,
     15, 14, 18, 15, 15, 13, 18, 14, 15, 15, 18, 22, 14, 22, 18, 17, 15,
     16, 15, 18, 15,  8, 22, 15, 15, 23])
```

```
y_pred = classifier.predict(X_test)
y_pred
```

```
array([14, 10, 14, 13, 14, 23, 16, 15, 23, 14, 15, 13, 16, 15, 18, 15, 18,
       14, 15, 23, 15, 13, 13, 15, 14, 23, 15, 22, 16, 13, 14, 23, 15, 16,
       18, 14, 16, 15, 23, 16, 22, 13, 14, 14, 22, 15, 15, 14, 14, 14, 14,
       16, 15, 14, 16, 23, 14, 15, 13, 22, 13, 14, 15, 15, 16, 15, 16, 14,
       15, 14, 16, 15, 15, 14, 15, 16, 15, 14, 15, 23, 14, 14, 14, 14, 18,
       14, 14, 15, 14, 23, 13, 13, 13, 15, 22, 15, 22, 15, 18, 15, 22, 14,
       16, 16, 13, 15, 14, 23, 14, 15, 13, 14,  0, 22, 22, 23, 15, 14, 16,
       14, 14, 14, 22, 14, 14, 15, 15, 22, 16, 13, 22,  8, 14, 15, 18, 14,
       22, 14, 15, 14, 15, 14, 14, 14, 23, 14, 14, 22, 15, 22, 23, 22, 18,
       16, 15, 22, 18, 15, 22, 13, 22, 13])
```

12.11

12-12

```
from sklearn.preprocessing import StandardScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.preprocessing import StandardScaler
scaler = MinMaxScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
/home/nick/anaconda3/lib/python3.7/site-packages/sklearn/preprocessing/data.py:323: DataConversionWarning: Data w
ith input dtype int64 were all converted to float64 by MinMaxScaler.
  return self.partial_fit(X, y)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
x=classifier.fit(X_train, y_train)
```

12-13

```python
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=5)
x=classifier.fit(X_train, y_train)
```

```python
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  2  2  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  7 23  0  0  3  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  2 38  0  1  0  0  1  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  2  3 10  0  0  0  4  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  8  1  0  8  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92         6
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         1
           7       0.00      0.00      0.00         2
           8       0.60      1.00      0.75         3
           9       0.00      0.00      0.00         0
          10       0.75      1.00      0.86         3
          12       0.00      0.00      0.00         1
          13       0.14      0.33      0.20         6
          14       0.61      0.70      0.65        33
          15       0.90      0.90      0.90        42
          16       0.83      0.50      0.62        20
          18       0.62      0.44      0.52        18
          19       0.00      0.00      0.00         1
          20       0.00      0.00      0.00         1
          22       0.72      0.93      0.81        14
          23       0.86      1.00      0.92         6
          24       0.00      0.00      0.00         1

   micro avg       0.69      0.69      0.69       162
   macro avg       0.33      0.37      0.34       162
weighted avg       0.69      0.69      0.68       162
```

```
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[ 1  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  0  1  0  0  0  0  0  3  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  1  0  0  1  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  3  7  1  0  0  0  2  1  0]
 [ 0  0  0  0  0  0  0  0  7 25  0  0  3  0  1  0  0]
 [ 0  0  0  0  0  0  0  0  2  1 30  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  1  3 13  2  0  2  2  0]
 [ 0  0  0  0  0  0  0  0  1  5  1  0  3  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  3  0  0  9  4  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  1  0  0  4  4  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0]]
              precision    recall  f1-score   support

           0       1.00      0.33      0.50         3
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           8       1.00      0.25      0.40         4
           9       0.00      0.00      0.00         2
          10       1.00      0.33      0.50         3
          12       0.00      0.00      0.00         1
          13       0.19      0.21      0.20        14
          14       0.53      0.69      0.60        36
          15       0.77      0.88      0.82        34
          16       0.76      0.57      0.65        23
          18       0.38      0.30      0.33        10
          19       0.00      0.00      0.00         1
          22       0.47      0.53      0.50        17
          23       0.31      0.40      0.35        10
          24       0.00      0.00      0.00         1

   micro avg       0.56      0.56      0.56       162
   macro avg       0.38      0.26      0.29       162
weighted avg       0.56      0.56      0.54       162
```

```
[[ 6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  2  2  0  2  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  7 23  0  0  3  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  2 38  0  1  0  0  1  0  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  2  3 10  0  0  0  4  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  8  1  0  8  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0 13  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0]]
```

	precision	recall	f1-score	support
0	0.86	1.00	0.92	6
1	0.00	0.00	0.00	1
2	0.00	0.00	0.00	1
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	1
6	0.00	0.00	0.00	1
7	0.00	0.00	0.00	2
8	0.60	1.00	0.75	3
9	0.00	0.00	0.00	0
10	0.75	1.00	0.86	3
12	0.00	0.00	0.00	1
13	0.14	0.33	0.20	6
14	0.61	0.70	0.65	33
15	0.90	0.90	0.90	42
16	0.83	0.50	0.62	20
18	0.62	0.44	0.52	18
19	0.00	0.00	0.00	1
20	0.00	0.00	0.00	1
22	0.72	0.93	0.81	14
23	0.86	1.00	0.92	6
24	0.00	0.00	0.00	1
micro avg	0.69	0.69	0.69	162
macro avg	0.33	0.37	0.34	162
weighted avg	0.69	0.69	0.68	162

```
[[ 6  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  1  0  0  0  1  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  3  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  2  2  0  2  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  7 23  0  0  3  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  2 38  0  1  0  0  1  0]
 [ 0  1  0  0  0  0  0  0  0  0  0  0  2  3 10  0  0  0  4  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  8  1  0  8  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0 13  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  6  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0]]
              precision    recall  f1-score   support

           0       0.86      1.00      0.92         6
           1       0.00      0.00      0.00         1
           2       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         1
           4       0.00      0.00      0.00         1
           6       0.00      0.00      0.00         1
           7       0.00      0.00      0.00         2
           8       0.60      1.00      0.75         3
           9       0.00      0.00      0.00         0
          10       0.75      1.00      0.86         3
          12       0.00      0.00      0.00         1
          13       0.14      0.33      0.20         6
          14       0.61      0.70      0.65        33
          15       0.90      0.90      0.90        42
          16       0.83      0.50      0.62        20
          18       0.62      0.44      0.52        18
          19       0.00      0.00      0.00         1
          20       0.00      0.00      0.00         1
          22       0.72      0.93      0.81        14
          23       0.86      1.00      0.92         6
          24       0.00      0.00      0.00         1

   micro avg       0.69      0.69      0.69       162
   macro avg       0.33      0.37      0.34       162
weighted avg       0.69      0.69      0.68       162
```

12-15

KNeighborsClassifier(*n_neighbors=5*, *weights='uniform'*, *algorithm='auto'*, *leaf_size=30*, *p=2*, *metric='minkowski'*, *metric_params=None*, *n_jobs=None*, ***kwargs*)

Weights str or callable, optional (default = ‘uniform’)

weight function used in prediction. Possible values:

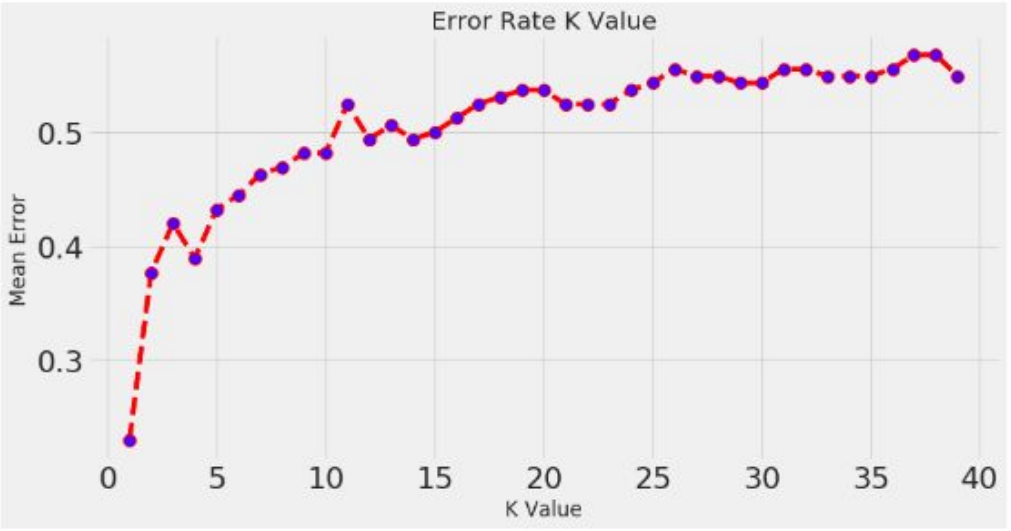
- **‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.**

- **‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.**
- **[callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.**

با

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Text(0,0.5,'Mean Error')
```



12-16

algorithm{‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’}, optional

Algorithm used to compute the nearest neighbors:

- **‘ball_tree’ will use BallTree**
- **‘kd_tree’ will use KDTree**
- **‘brute’ will use a brute-force search.**
- **‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method.**

Note: fitting on sparse input will override the setting of this parameter, using brute force.


```
train_accuracy = []

# Calculating error for K values between 1 and 40
for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_train)
    train_accuracy.append(np.mean(pred_i != y_train))

train_accuracy
```

```
[0.014492753623188406,
 0.09316770186335403,
 0.13043478260869565,
 0.17598343685300208,
 0.2111801242236025,
 0.2277432712215321,
 0.2443064182194617,
 0.2505175983436853,
 0.27536231884057971]
```

```
test_accuracy = []

# Calculating error for K values between 1 and 40
for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    test_accuracy.append(np.mean(pred_i != y_test))

test_accuracy
```

```
[0.19135802469135801,
 0.2777777777777778,
 0.345679012345679,
 0.32098765432098764,
 0.35802469135802467,
 0.37037037037037035,
 0.35802469135802467,
 0.37037037037037035,
 0.36419753086419754]
```

```
train_accuracy = []

# Calculating error for K values between 1 and 40
for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_train)
    train_accuracy.append(np.mean(pred_i != y_train))

train_accuracy
```

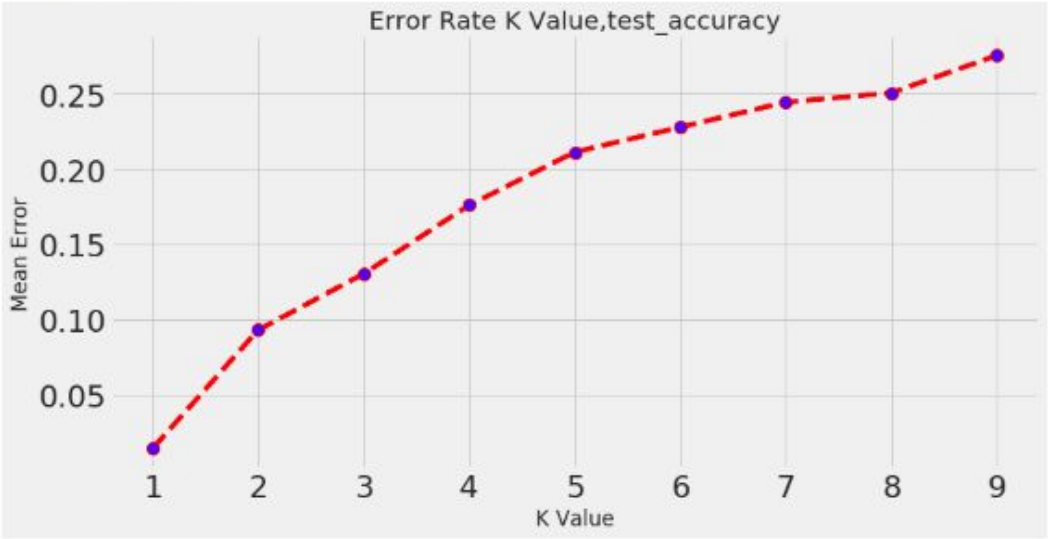
```
[0.09109730848861283,
 0.16977225672877846,
 0.20496894409937888,
 0.2732919254658385,
 0.29606625258799174,
 0.3333333333333333,
 0.35403726708074534,
 0.37267080745341613,
 0.391304347826087]
```

```
test_accuracy = []

# Calculating error for K values between 1 and 40
for i in range(1, 10):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    test_accuracy.append(np.mean(pred_i != y_test))

test_accuracy
```

```
[0.22839506172839505,
 0.3765432098765432,
 0.41975308641975306,
 0.3888888888888889,
 0.43209876543209874,
 0.4444444444444444,
 0.46296296296296297,
 0.4691358024691358,
 0.48148148148148145]
```

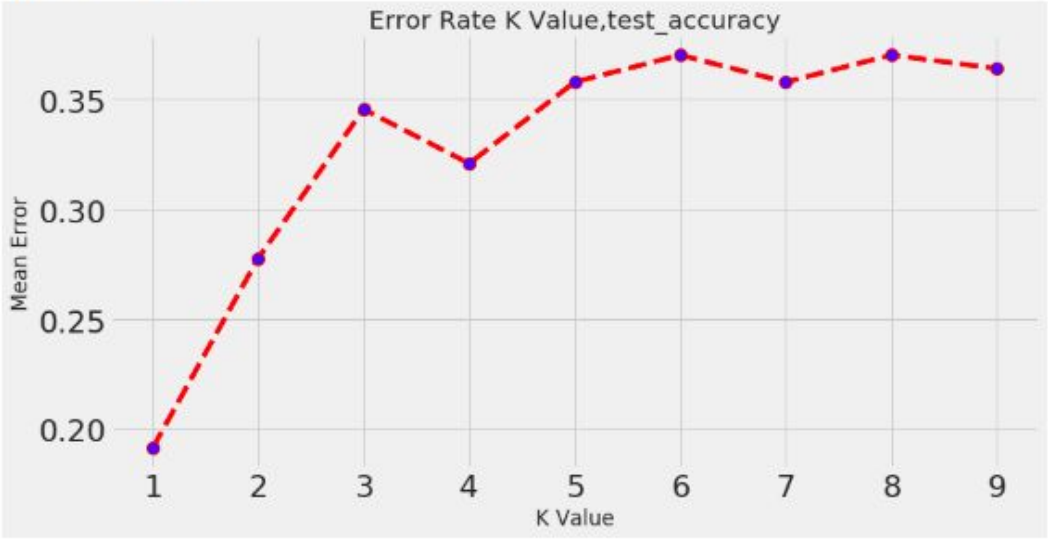
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), train_accuracy, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value,test_accuracy')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Text(0,0.5,'Mean Error')
```



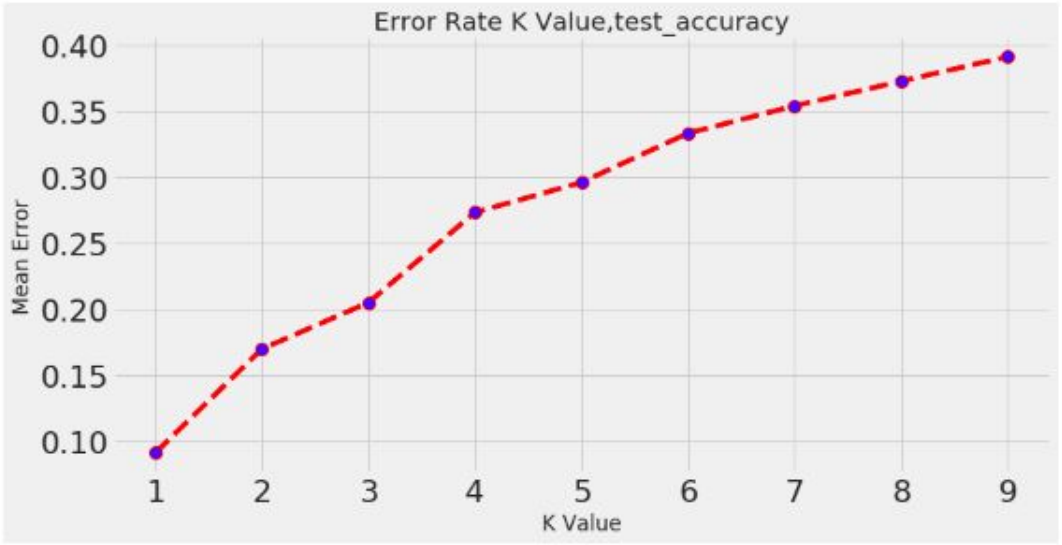

```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), test_accuracy, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value,test_accuracy')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Text(0,0.5,'Mean Error')
```



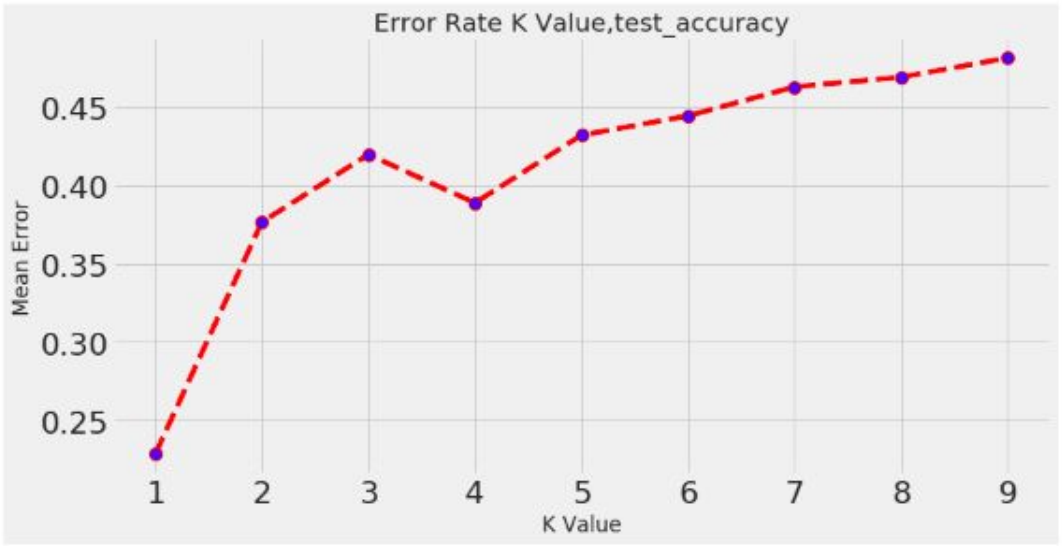
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), train_accuracy, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value,test_accuracy')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Text(0,0.5,'Mean Error')
```




```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 10), test_accuracy, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate K Value,test_accuracy')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

```
Text(0,0.5,'Mean Error')
```



نمودار بالایی ترین هست و پایینی تست

12-19

نشان می دهد خطا در هر دو رو به افزایش است و طبق نمودار درتست بدتر هم هست نشان می دهد که مدل با این حال شاید انتخاب k ۴ مناسب یاشد