UNIT-II

Introduction

Data Manipulation is an important phase of predictive modeling. A robust predictive model cannot be built using machine learning algorithms. But, with an approach to understand the business problem, the underlying data and extracting business insights are done performing required data manipulations. Among several phases of model building, most of the time is usually spent in understanding underlying data and performing required manipulations.

Data Manipulation

It involves 'manipulating' data using available set of variables. This is done to enhance accuracy and precision associated with data. Actually, the data collection process can have many loopholes. There are various uncontrollable factors which lead to inaccuracy in data such as mental situation of respondents, personal biases, difference / error in readings of machines etc. To lessen these inaccuracies, data manipulation is done to increase the possible (highest) accuracy in data. This stage is also known as data wrangling or data cleaning.

Different Ways to Manipulate / Treat Data:

Manipulating data using inbuilt base R functions. This is the first step, but is often repetitive and time consuming. Hence, it is a less efficient way to solve the problem.
Use of packages for data manipulation. CRAN has more than 8000 packages available today. These packages are a collection of pre-written commonly used pieces of codes. They helps to perform the repetitive tasks fasts, reduce errors in coding and take help of code written by experts (across the open source eco-system for R) to make code more efficient. This is usually the most common way of performing data manipulation.
Use of Machine Learning(ML) algorithms for data manipulation. ML algorithms like tree based boosting algorithms to take care of missing data & outliers. These algorithms are less time consuming,

Note: Install packages using:

```
install.packages('package name')
```

List of Packages

1. dplyr
2. data.table
3. ggplot2
4. reshape2
5. readr
6. tidyr

7. lubridate

dplyr Package

This package is created and maintained by Hadley Wickham. This package has everything (almost) to accelerate data manipulation efforts. It is known best for data exploration and transformation. Its chaining syntax makes it highly adaptive to use. It includes 5 major data manipulation commands:

1. filter – It filters the data based on a condition
2. select – It is used to select columns of interest from a data set
3. arrange – It is used to arrange data set values on ascending or descending order
4. mutate – It is used to create new variables from existing variables
5. summarise (with group_by) – It is used to perform analysis by commonly used operations such as min, max, mean count etc

Note : 2 pre-installed R data sets namely mtcars and iris.

```
> library(dplyr)

> data("mtcars")
> data('iris')

> mydata <- mtcars
#read data
> head(mydata)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

```
#creating a local dataframe.

Local data frame are easier to read

> mynewdata <- tbl_df(mydata)

> myirisdata <- tbl_df(iris)

#now data will be in tabular structure

> mynewdata
```

```
Source: local data frame [32 x 11]

     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
..   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
```

> myirisdata	

```
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          (dbl)       (dbl)        (dbl)       (dbl)  (fctr)
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
..          ...         ...          ...         ...     ...
```

#use filter to filter data with required condition

> filter(mynewdata, cyl > 4 & gear > 4)

```
Source: local data frame [3 x 11]

    mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
  (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
1  15.8     8   351   264  4.22  3.17  14.5     0     1     5     4
2  19.7     6   145   175  3.62  2.77  15.5     0     1     5     6
3  15.0     8   301   335  3.54  3.57  14.6     0     1     5     8
```

> filter(mynewdata, cyl > 4)

```
Source: local data frame [21 x 11]

     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
4   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
5   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
6   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
7   19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
8   17.8     6 167.6   123  3.92 3.440 18.90     1     0     4     4
9   16.4     8 275.8   180  3.07 4.070 17.40     0     0     3     3
10  17.3     8 275.8   180  3.07 3.730 17.60     0     0     3     3
..   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
```

> filter(myirisdata, Species %in% c('setosa', 'virginica'))

```
Source: local data frame [100 x 5]

   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
          (dbl)       (dbl)        (dbl)       (dbl)  (fctr)
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa
..          ...         ...          ...         ...     ...
```

#use select to pick columns by name

> select(mynewdata, cyl,mpg,hp)

```
      cyl   mpg    hp
    (dbl) (dbl) (dbl)
1       6  21.0   110
2       6  21.0   110
3       4  22.8    93
4       6  21.4   110
5       8  18.7   175
6       6  18.1   105
7       8  14.3   245
8       4  24.4    62
9       4  22.8    95
10      6  19.2   123
..    ...   ...   ...
```

#here you can use (-) to hide columns

> select(mynewdata, -cyl, -mpg)

	disp (dbl)	hp (dbl)	drat (dbl)	wt (dbl)	qsec (dbl)	vs (dbl)	am (dbl)	gear (dbl)	carb (dbl)
1	160.0	110	3.90	2.620	16.46	0	1	4	4
2	160.0	110	3.90	2.875	17.02	0	1	4	4
3	108.0	93	3.85	2.320	18.61	1	1	4	1
4	258.0	110	3.08	3.215	19.44	1	0	3	1
5	360.0	175	3.15	3.440	17.02	0	0	3	2
6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	360.0	245	3.21	3.570	15.84	0	0	3	4
8	146.7	62	3.69	3.190	20.00	1	0	4	2
9	140.8	95	3.92	3.150	22.90	1	0	4	2
10	167.6	123	3.92	3.440	18.30	1	0	4	4
..	...	...	...	...	...	...	...	...	...

#hide a range of columns

> select(mynewdata, -c(cyl,mpg))

	disp (dbl)	hp (dbl)	drat (dbl)	wt (dbl)	qsec (dbl)	vs (dbl)	am (dbl)	gear (dbl)	carb (dbl)
1	160.0	110	3.90	2.620	16.46	0	1	4	4
2	160.0	110	3.90	2.875	17.02	0	1	4	4
3	108.0	93	3.85	2.320	18.61	1	1	4	1
4	258.0	110	3.08	3.215	19.44	1	0	3	1
5	360.0	175	3.15	3.440	17.02	0	0	3	2
6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	360.0	245	3.21	3.570	15.84	0	0	3	4
8	146.7	62	3.69	3.190	20.00	1	0	4	2
9	140.8	95	3.92	3.150	22.90	1	0	4	2
10	167.6	123	3.92	3.440	18.30	1	0	4	4
..	...	...	...	...	...	...	...	...	...

#select series of columns

> select(mynewdata, cyl:gear)

	cyl (dbl)	disp (dbl)	hp (dbl)	drat (dbl)	wt (dbl)	qsec (dbl)	vs (dbl)	am (dbl)	gear (dbl)
1	6	160.0	110	3.90	2.620	16.46	0	1	4
2	6	160.0	110	3.90	2.875	17.02	0	1	4
3	4	108.0	93	3.85	2.320	18.61	1	1	4
4	6	258.0	110	3.08	3.215	19.44	1	0	3
5	8	360.0	175	3.15	3.440	17.02	0	0	3
6	6	225.0	105	2.76	3.460	20.22	1	0	3
7	8	360.0	245	3.21	3.570	15.84	0	0	3
8	4	146.7	62	3.69	3.190	20.00	1	0	4
9	4	140.8	95	3.92	3.150	22.90	1	0	4
10	6	167.6	123	3.92	3.440	18.30	1	0	4
..	...	...	...	...	...	...	...	...	...

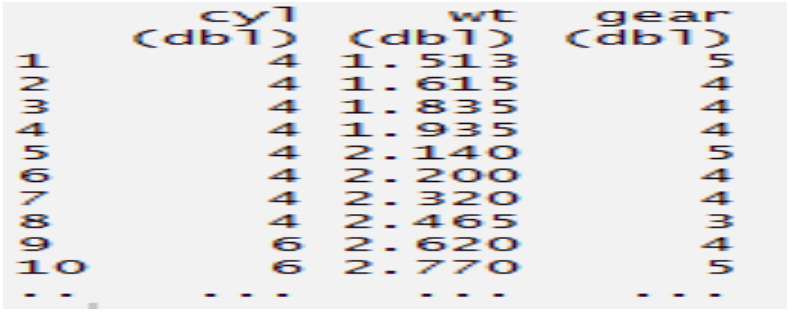
#chaining or pipelining - a way to perform multiple operations #in one line

> mynewdata %>% select(cyl, wt, gear)%>% filter(wt > 2)

	cyl (dbl)	wt (dbl)	gear (dbl)
1	6	2.620	4
2	6	2.875	4
3	4	2.320	4
4	6	3.215	3
5	8	3.440	3
6	6	3.460	3
7	8	3.570	3
8	4	3.190	4
9	4	3.150	4
10	6	3.440	4
..	...	...	...

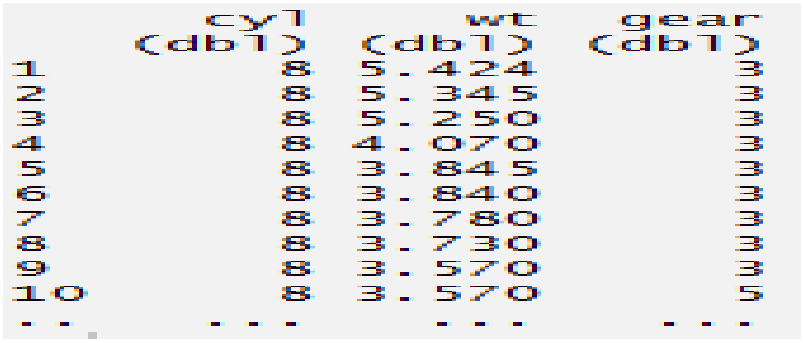
#arrange can be used to reorder rows

> mynewdata%>% select(cyl, wt, gear)%>% arrange(wt)



	cyl	wt	gear
	(dbl)	(dbl)	(dbl)
1	4	1.513	5
2	4	1.615	4
3	4	1.835	4
4	4	1.935	4
5	4	2.140	5
6	4	2.200	4
7	4	2.320	4
8	4	2.465	3
9	6	2.620	4
10	6	2.770	5
..	...	...	...

> mynewdata%>% select(cyl, wt, gear)%>% arrange(desc(wt))

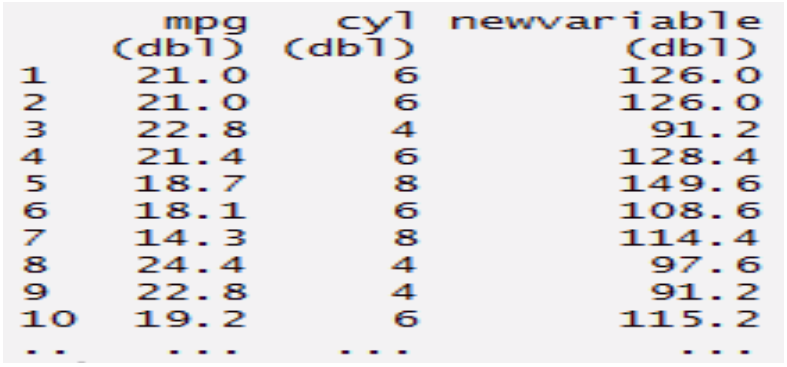


	cyl	wt	gear
	(dbl)	(dbl)	(dbl)
1	8	5.424	3
2	8	5.345	3
3	8	5.250	3
4	8	4.070	3
5	8	3.845	3
6	8	3.840	3
7	8	3.780	3
8	8	3.730	3
9	8	3.570	3
10	8	3.570	5
..	...	...	...

#mutate - create new variables

> mynewdata %>% select(mpg, cyl)%>% mutate(newvariable = mpg*cyl)

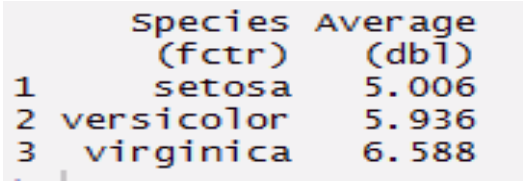
> newvariable <- mynewdata %>% mutate(newvariable = mpg*cyl)



	mpg	cyl	newvariable
	(dbl)	(dbl)	(dbl)
1	21.0	6	126.0
2	21.0	6	126.0
3	22.8	4	91.2
4	21.4	6	128.4
5	18.7	8	149.6
6	18.1	6	108.6
7	14.3	8	114.4
8	24.4	4	97.6
9	22.8	4	91.2
10	19.2	6	115.2
..	...	...	...

#summarise - this is used to find insights from data

> myirisdata%>% group_by(Species)%>% summarise(Average = mean(Sepal.Length, na.rm = TRUE))



	Species	Average
	(fctr)	(dbl)
1	setosa	5.006
2	versicolor	5.936
3	virginica	6.588

#summarise each

> myirisdata%>% group_by(Species)%>% summarise_each(funs(mean, n()), Sepal.Length, Sepal.Width)

Species	Sepal.Length_mean	Sepal.Width_mean	Sepal.Length_n	Sepal.Width_n
(fctr)	(dbl)	(dbl)	(int)	(int)
1 setosa	5.006	3.428	50	50
2 versicolor	5.936	2.770	50	50
3 virginica	6.588	2.974	50	50

rename the variables using rename command

> mynewdata %>% rename(miles = mpg)

	miles	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)	(dbl)
1	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
2	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
3	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
4	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
5	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
6	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
7	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
8	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
9	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
10	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
..	...	...	...	...	...	...	...	...	...	...	...

data.table Package

This package allows to perform faster manipulation in a data set. A data table has 3 parts namely DT[i,j,by]. We can tell R to subset the rows using ‘i’, to calculate ‘j’ which is grouped by ‘by’. Most of the times, ‘by’ relates to categorical variable.

#load data

> data("airquality")
> mydata <- airquality
> head(airquality,6)

	Ozone	Solar.R	Wind	Temp	Month	Day
1	41	190	7.4	67	5	1
2	36	118	8.0	72	5	2
3	12	149	12.6	74	5	3
4	18	313	11.5	62	5	4
5	NA	NA	14.3	56	5	5
6	28	NA	14.9	66	5	6

#load package

> library(data.table)

> mydata <- data.table(mydata)
> mydata

```
      Ozone Solar.R Wind Temp Month Day
  1:     41     190  7.4   67     5   1
  2:     36     118  8.0   72     5   2
  3:     12     149 12.6   74     5   3
  4:     18     313 11.5   62     5   4
  5:     NA      NA 14.3   56     5   5
 ---
149:     30     193  6.9   70     9  26
150:     NA     145 13.2   77     9  27
151:     14     191 14.3   75     9  28
152:     18     131  8.0   76     9  29
153:     20     223 11.5   68     9  30
```

> myiris <- data.table(myiris)

> myiris

```
     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
  1:          5.1         3.5          1.4         0.2    setosa
  2:          4.9         3.0          1.4         0.2    setosa
  3:          4.7         3.2          1.3         0.2    setosa
  4:          4.6         3.1          1.5         0.2    setosa
  5:          5.0         3.6          1.4         0.2    setosa
 ---
146:          6.7         3.0          5.2         2.3 virginica
147:          6.3         2.5          5.0         1.9 virginica
148:          6.5         3.0          5.2         2.0 virginica
149:          6.2         3.4          5.4         2.3 virginica
150:          5.9         3.0          5.1         1.8 virginica
```

#subset rows - select 2nd to 4th row

> mydata[2:4,]

```
   Ozone Solar.R Wind Temp Month Day
1:    36     118  8.0   72     5   2
2:    12     149 12.6   74     5   3
3:    18     313 11.5   62     5   4
```

#select columns with particular values

> myiris[Species == 'setosa']

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1:	5.1	3.5	1.4	0.2	setosa
2:	4.9	3.0	1.4	0.2	setosa
3:	4.7	3.2	1.3	0.2	setosa
4:	4.6	3.1	1.5	0.2	setosa
5:	5.0	3.6	1.4	0.2	setosa
6:	5.4	3.9	1.7	0.4	setosa
7:	4.6	3.4	1.4	0.3	setosa
8:	5.0	3.4	1.5	0.2	setosa
9:	4.4	2.9	1.4	0.2	setosa
10:	4.9	3.1	1.5	0.1	setosa
11:	5.4	3.7	1.5	0.2	setosa

#select columns with multiple values. This will give you columns with Setosa #and virginica species

> myiris[Species %in% c('setosa', 'virginica')]

#select columns. Returns a vector

> mydata[,Temp]

```
  [1] 67 72 74 62 56 66 65 59 61 69 74 69 66 68 58 64 66 57 68 62 59 73 61 61 57 58 57
 [28] 67 81 79 76 78 74 67 84 85 79 82 87 90 87 93 92 82 80 79 77 72 65 73 76 77 76 76
 [55] 76 75 78 73 80 77 83 84 85 81 84 83 83 88 92 92 89 82 73 81 91 80 81 82 84 87 85
 [82] 74 81 82 86 85 82 86 88 86 83 81 81 81 82 86 85 87 89 90 90 92 86 86 82 80 79 77
[109] 79 76 78 78 77 72 75 79 81 86 88 97 94 96 94 91 92 93 93 87 84 80 78 75 73 81 76
[136] 77 71 71 78 67 76 68 82 64 71 81 69 63 70 77 75 76 68
```

> mydata[,.(Temp,Month)]

	Temp	Month
1:	67	5
2:	72	5
3:	74	5
4:	62	5
5:	56	5

149:	70	9
150:	77	9
151:	75	9
152:	76	9
153:	68	9

#returns sum of selected column

> mydata[,sum(Ozone, na.rm = TRUE)]

[1]4887

#returns sum and standard deviation

> mydata[,.(sum(Ozone, na.rm = TRUE), sd(Ozone, na.rm = TRUE))]

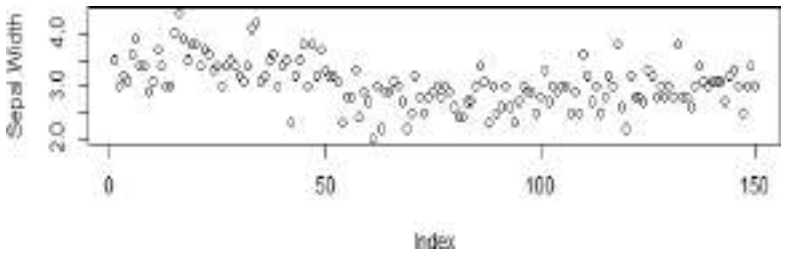
```
     V1       V2
1: 4887 32.98788
```

#print and plot

> myiris[,{print(Sepal.Length)

> plot(Sepal.Width) NULL}]

```
  [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1
 [21] 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1
 [41] 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3 5.0 7.0 6.4 6.9 5.5 6.5 5.7 6.3 4.9 6.6 5.2
 [61] 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6 6.8 6.7 6.0 5.7
 [81] 5.5 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7
[101] 6.3 5.8 7.1 6.3 6.5 7.6 4.9 7.3 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0
[121] 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2 7.4 7.9 6.4 6.3 6.1 7.7 6.3 6.4 6.0 6.9
[141] 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```



#grouping by a variable

> myiris[,.(sepalsum = sum(Sepal.Length)), by=Species

```
      Species sepalsum
1:     setosa    250.3
2: versicolor    296.8
3:  virginica    329.4
```

#select a column for computation, hence need to set the key on column

> setkey(myiris, Species)

#selects all the rows associated with this data point

> myiris['setosa']
> myiris[c('setosa', 'virginica')]

ggplot2 Package

ggplot offers a whole new world of colors and patterns. Plotting 3 graphs: Scatter Plot, Bar Plot, Histogram. ggplot is enriched with customized features to make visualization better. It becomes even more powerful when grouped with other packages like cowplot, gridExtra.

Scatter Plot :

A Scatter Plot is a graph in which the values of two variables are plotted along two axes, the pattern of the resulting points revealing any correlation present.

With scatter plots we can explain how the variables relate to each other. Which is defined as correlation. Positive, Negative, and None (no correlation) are the three types of correlation.

Limitations of a Scatter Diagram

Below are the few limitations of a scatter diagram:

• With Scatter diagrams we cannot get the exact extent of correlation.

• Quantitative measure of the relationship between the variable cannot be viewed. Only shows the quantitative expression.

• The relationship can only show for two variables.

Advantages of a Scatter Diagram

Below are the few advantages of a scatter diagram:

• Relationship between two variables can be viewed.

• For non-linear pattern, this is the best method.

• Maximum and minimum value, can be easily determined.

• Observation and reading is easy to understand

• Plotting the diagram is very simple.

Bar Plot

A barplot (or barchart) is one of the most common type of graphic. It shows the relationship between a numeric variable and a categoric variable.

Bar Plot are classified into four types of graphs - bar graph or bar chart, line graph, pie chart, and diagram.

Limitations of Bar Plot:

When we try to display changes in speeds such as acceleration, Bar graphs wont help us.

Advantages of Bar plot:

• Bar charts are easy to understand and interpret.

• Relationship between size and value helps for in easy comparison.

• They're simple to create.

• They can help in presenting very large or very small values easily.

Histogram

A histogram represents the frequency distribution of continuous variables. while, a bar

graph is a diagrammatic comparison of discrete variables.

Histogram presents numerical data whereas bar graph shows categorical data. The histogram is drawn in such a way that there is no gap between the bars.

Limitations of Histogram:

A histogram can present data that is misleading as it has many bars.

Only two sets of data are used, but to analyze certain types of statistical data, more than two sets of data are necessary

Advantages of Histogram:

Histogram helps to identify different data, the frequency of the data occurring in the dataset and categories which are difficult to interpret in a tabular form. It helps to visualize the distribution of the data.

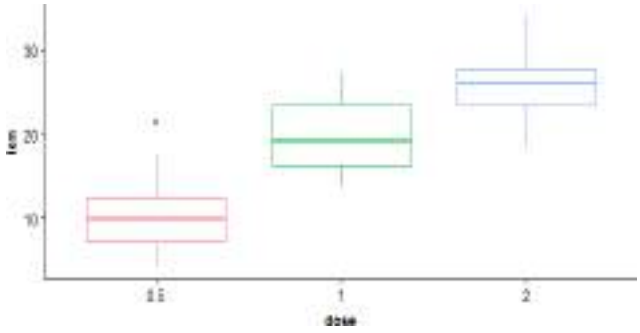
```
> library(ggplot2)
> library(gridExtra)
> df <- ToothGrowth
> df$dose <- as.factor(df$dose)
> head(df)
```

	len	supp	dose
1	4.2	VC	0.5
2	11.5	VC	0.5
3	7.3	VC	0.5
4	5.8	VC	0.5
5	6.4	VC	0.5
6	10.0	VC	0.5

BOX PLOT

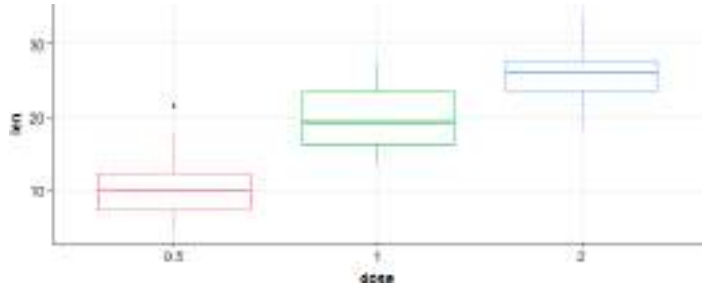
```
> bp <- ggplot(df, aes(x = dose, y = len, color = dose)) + geom_boxplot() +
theme(legend.position = 'none')

> bp
```



#add gridlines

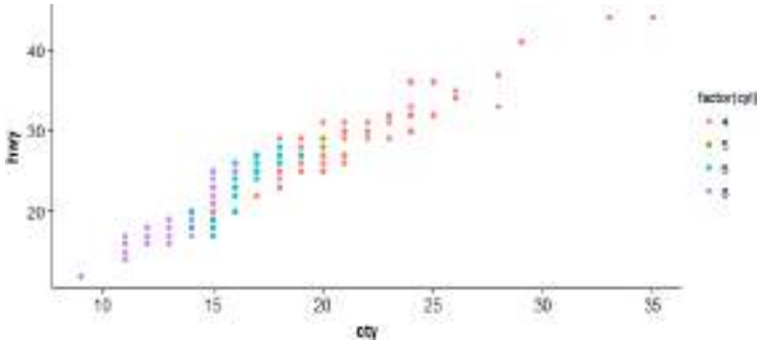
> bp + background_grid(major = "xy", minor = 'none')



SCATTER PLOT

> sp <- ggplot(mpg, aes(x = cty, y = hwy, color = factor(cyl)))+geom_point(size = 2.5)

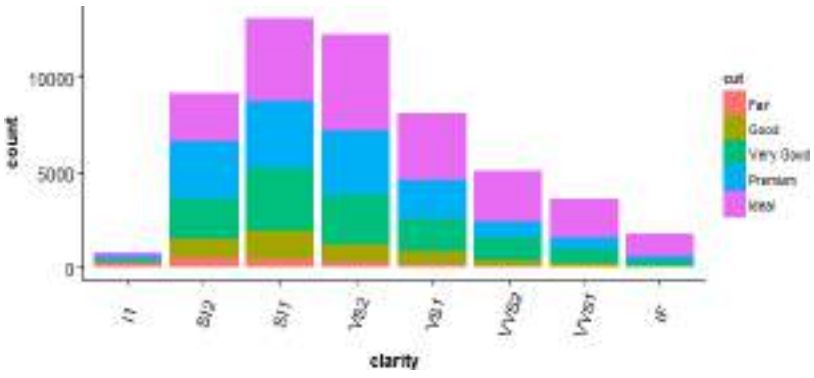
> sp



BAR PLOT

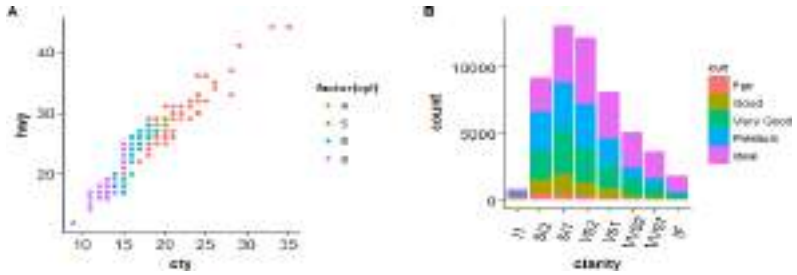
> bp <- ggplot(diamonds, aes(clarity, fill = cut)) + geom_bar() +theme(axis.text.x = element_text(angle = 70, vjust = 0.5))

> bp



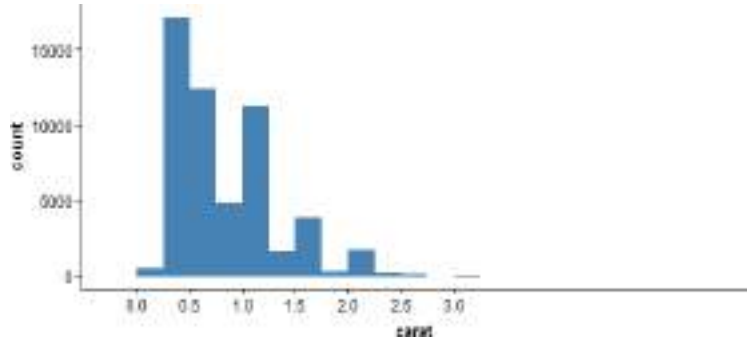
#compare two plots

> plot_grid(sp, bp, labels = c("A","B"), ncol = 2, nrow = 1)



#histogram

> ggplot(diamonds, aes(x = carat)) + geom_histogram(binwidth = 0.25, fill = 'steelblue')+scale_x_continuous(breaks=seq(0,3, by=0.5))



reshape2 Package

As the name suggests, this package is useful in reshaping data. The data come in many forms. Hence, we are required to shape it according to our need. Usually, the process of reshaping data in R is tedious. R base functions consist of 'Aggregation' option using which data can be reduced and rearranged into smaller forms, but with reduction in amount of information. Aggregation includes tapply, by and aggregate base functions. The reshape package overcomes these problems. It has 2 functions namely melt and cast.

melt : This function converts data from wide format to long format. It's a form of restructuring where multiple categorical columns are 'melted' into unique rows.

#create a data

> ID <- c(1,2,3,4,5)

> Names <- c('Joseph','Matrin','Joseph','James','Matrin')

> DateofBirth <- c(1993,1992,1993,1994,1992)

> Subject<- c('Maths','Biology','Science','Psycology','Physics')

> thisdata <- data.frame(ID, Names, DateofBirth, Subject)

> data.table(thisdata)

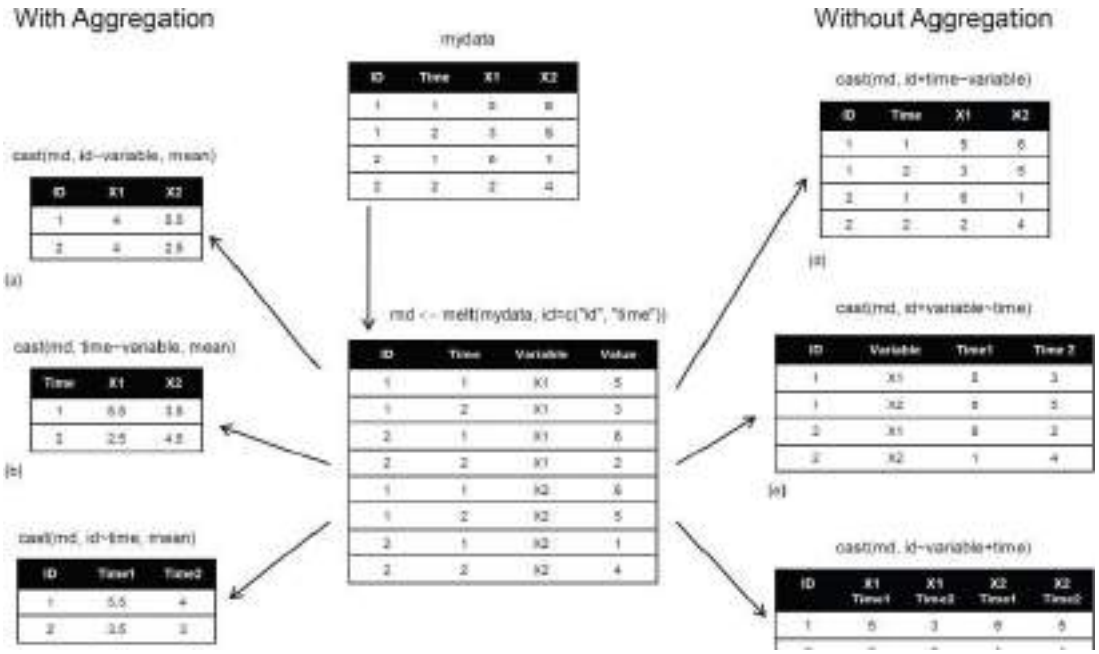
```
   ID  Names DateofBirth   Subject
1:  1 Joseph        1993     Maths
2:  2 Matrin        1992   Biology
3:  3 Joseph        1993   Science
4:  4  James        1994 Psycology
5:  5 Matrin        1992   Physics
```

#load package
> install.packages('reshape2')
> library(reshape2)
#melt
> mt <- melt(thisdata, id=(c('ID','Names')))
> mt

	ID	Names	variable	value
1	1	Joseph	DateofBirth	1993
2	2	Matrin	DateofBirth	1992
3	3	Joseph	DateofBirth	1993
4	4	James	DateofBirth	1994
5	5	Matrin	DateofBirth	1992
6	1	Joseph	Subject	Maths
7	2	Matrin	Subject	Biology
8	3	Joseph	Subject	Science
9	4	James	Subject	Psycology
10	5	Matrin	Subject	Physics

cast : This function converts data from long format to wide format. It starts with melted data and reshapes into long format. It's just the reverse of melt function. It has two functions namely, dcast and acast. dcast returns a data frame as output. acast returns a vector/matrix/array as the output.
> mcast <- dcast(mt, DateofBirth + Subject ~ variable)
> mcast

	DateofBirth	Subject	DateofBirth	Subject
1	1992	Biology	1992	Biology
2	1992	Physics	1992	Physics
3	1993	Maths	1993	Maths
4	1993	Science	1993	Science
5	1994	Psycology	1994	Psycology



tidyr Package

This package can make the data look 'tidy'. It has 4 major functions to accomplish this task. The 4 functions are:

gather() – it ‘gathers’ multiple columns. Then, it converts them into key:value pairs.
This function will transform wide from of data to long form. You can use it as in alternative to ‘melt’ in reshape package.
spread() – It does reverse of gather. It takes a key:value pair and converts it into separate columns.
separate() – It splits a column into multiple columns.

unite() – It does reverse of separate. It unites multiple columns into single column

```
#load package
> library(tidyr)
#create a dummy data set
> names <- c('A','B','C','D','E','A','B')
> weight <- c(55,49,76,71,65,44,34)
> age <- c(21,20,25,29,33,32,38)

> Class <- c('Maths','Science','Social','Physics','Biology','Economics','Accounts')

#create data frame
> tdata <- data.frame(names, age, weight, Class)

> tdata
```

	names	age	weight	Class
1	A	21	55	Maths
2	B	20	49	Science
3	C	25	76	Social
4	D	29	71	Physics
5	E	33	65	Biology
6	A	32	44	Economics
7	B	38	34	Accounts

#using gather function

```
> long_t <- tdata %>% gather(Key, Value, weight:Class)
> long_t
```

	names	age	Key	Value
1	A	21	weight	55
2	B	20	weight	49
3	C	25	weight	76
4	D	29	weight	71
5	E	33	weight	65
6	A	32	weight	44
7	B	38	weight	34
8	A	21	Class	Maths
9	B	20	Class	Science
10	C	25	Class	Social
11	D	29	Class	Physics
12	E	33	Class	Biology
13	A	32	Class	Economics
14	B	38	Class	Accounts

Separate Command

#create a data set

Time <- c("27/01/2015 15:44","23/02/2015 23:24", "31/03/2015 19:15", "20/01/2015 20:52", "23/02/2015 07:46", "31/01/2015 01:55")

#build a data frame

> d_set <- data.frame(Humidity, Rain, Time)

#using separate function we can separate date, month, year

> separate_d <- d_set %>% separate(Time, c('Date', 'Month','Year'))

> separate_d

	Humidity	Rain	Date	Month	Year
1	37.79	0.9713604	27	01	2015
2	42.34	1.1096972	23	02	2015
3	52.16	1.0644759	31	03	2015
4	44.57	0.9531834	20	01	2015
5	43.83	0.9887885	23	02	2015
6	44.59	0.9396761	31	01	2015

Unite Command

#using unite function - reverse of separate

> unite_d <- separate_d%>% unite(Time, c(Date, Month, Year), sep = "/")

> unite_d

	Humidity	Rain	Time
1	37.79	0.9713604	27/01/2015
2	42.34	1.1096972	23/02/2015
3	52.16	1.0644759	31/03/2015
4	44.57	0.9531834	20/01/2015
5	43.83	0.9887885	23/02/2015
6	44.59	0.9396761	31/01/2015

Spread Function (reverse of gather command)

#using spread function - reverse of gather

> wide_t <- long_t %>% spread(Key, Value)

> wide_t

	names	age	weight	Class
1	A	21	55	Maths
2	A	32	44	Economics
3	B	20	49	Science
4	B	38	34	Accounts
5	C	25	76	Social
6	D	29	71	Physics
7	E	33	65	Biology

readr Package

'readr' helps in reading various forms of data into R. With 10x faster speed. Here, characters are never converted to factors. This package can replace the traditional read.csv() and read.table() base R functions. It helps in reading the following data:

Delimited files with read_delim(), read_csv(), read_tsv(), and read_csv2().
Fixed width files with read_fwf(), and read_table().
Web log files with read_log()

If the data loading time is more than 5 seconds, this function will show you a progress bar too.

> install.packages('readr')	> library(readr)
	> read_csv('test.csv',col_names = TRUE)
specify the data type of every column loaded in data	> read_csv("iris.csv", col_types = list(Sepal.Length = col_double(), Sepal.Width = col_double(), Petal.Length = col_double(), Petal.Width = col_double(), Species = col_factor(c("setosa", "versicolor", "virginica"))))
choose to omit unimportant columns	> read_csv("iris.csv", col_types = list(Species = col_factor (c("setosa", "versicolor", "virginica")))

Lubridate Package

Lubridate package reduces the pain of working of data time variable in R. The inbuilt function of this package offers a nice way to make easy parsing in dates and times. This package is frequently used with data comprising of timely data.

> install.packages('lubridate')
> library(lubridate)

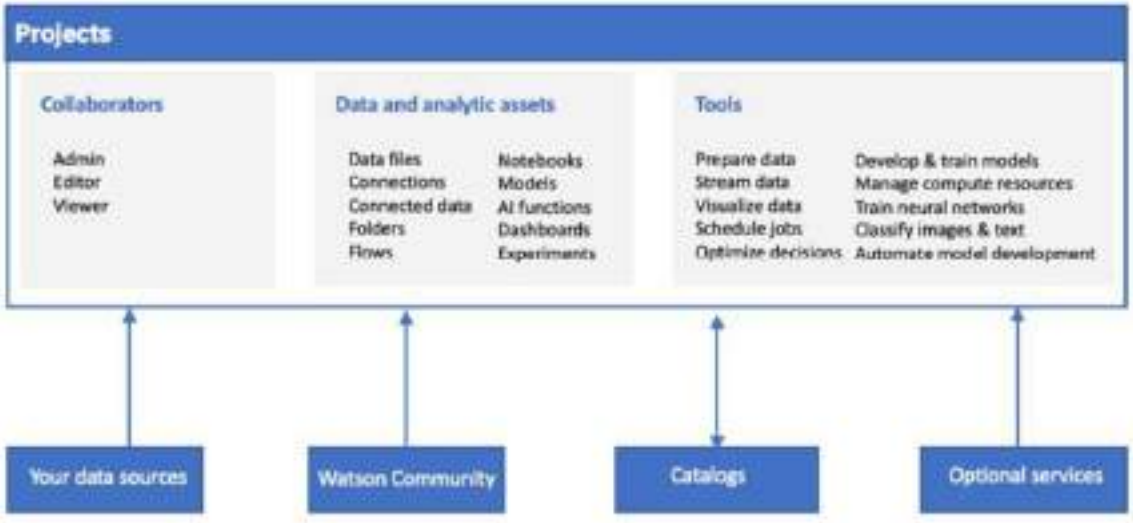
#current date and time	> now()	[1] "2015-12-11 13:23:48 IST"
#assigning current date and time to variable n_time	> n_time <- now()	
#using update function	>n_update<-update(n_time, year = 2013, month = 10) > n_update	[1] "2013-10-11 13:24:28 IST"

#add days, months, year, seconds	> d_time <- now() > d_time + ddays(1)	[1] "2015-12-12 13:24:54 IST"
	> d_time + dweeks(2)	[1] "2015-12-12 13:24:54 IST"
	> d_time + dyears(3)	[1] "2018-12-10 13:24:54 IST"
	> d_time + dhours(2)	[1] "2015-12-11 15:24:54 IST"
	> d_time + dminutes(50)	[1] "2015-12-11 14:14:54 IST"
	> d_time + dseconds(60)	[1] "2015-12-11 13:25:54 IST"
#extract date,time	> n_time$hour <- hour(now()) > n_time$minute <- minute(now()) > n_time$second <- second(now()) > n_time$month <- month(now()) > n_time$year <- year(now())	
#check the extracted dates in separate columns	> new_data <- data.frame(n_time$hour, n_time$minute, n_time$second, n_time$month, n_time$year) > new_data	n_time.hour n_time.minute n_time.second n_time.month 13 27 41.65723 12

WATSON STUDIO

Watson Studio provides you with the environment and tools to solve your business problems by collaboratively working with data. You can choose the tools you need to analyze and visualize data, to cleanse and shape data, to ingest streaming data, or to create and train machine learning models.

This illustration shows how the architecture of Watson Studio is centered around the project. A project is where you organize your resources and work with data.



Visualizing information in graphical ways can give you insights into your data. By enabling you to look at and explore data from different perspectives, visualizations can help you identify patterns, connections, and relationships within that data as well as understand large amounts of information very quickly.

Create a project -
To create a project :

Click New project on the Watson Studio home page or your My Projects page.

Choose whether to create an empty project or to create a project based on an exported project file or a sample project.

If you chose to create a project from a file or a sample, upload a project file or select a sample project. See Importing a project.

On the New project screen, add a name and optional description for the

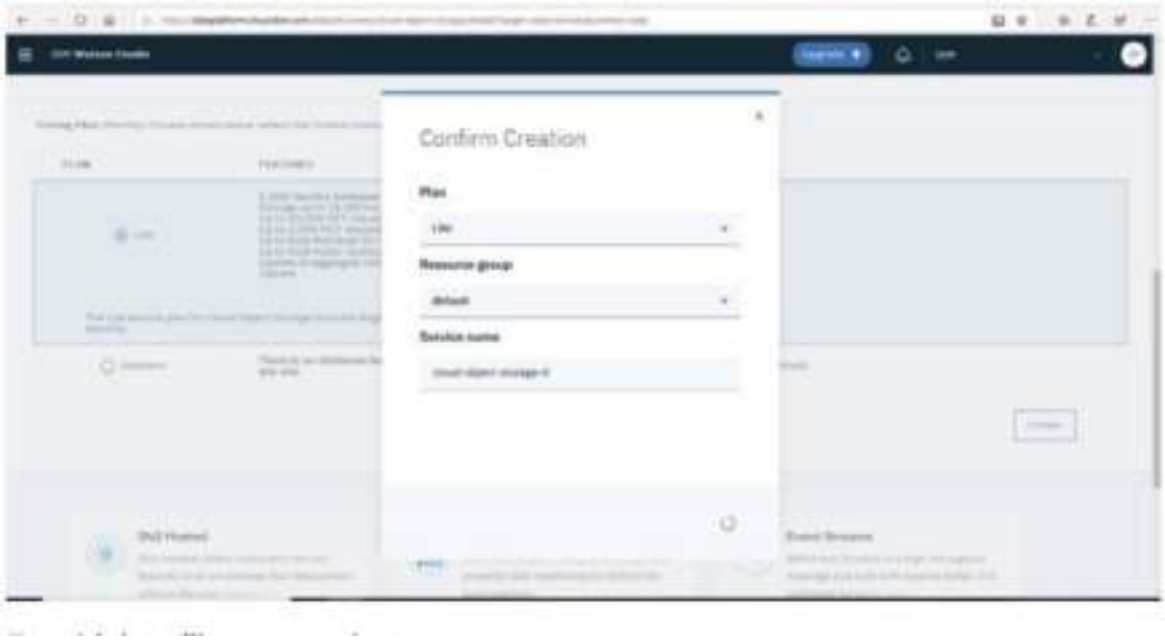
project.

Select the Restrict who can be a collaborator check box to restrict collaborators to members of your organization or integrate with a catalog. The check box is selected by default if you are a member of a catalog. You can't change this setting after you create the project.

If prompted, choose or add any required services.

Choose an existing object storage service instance or create a new one.

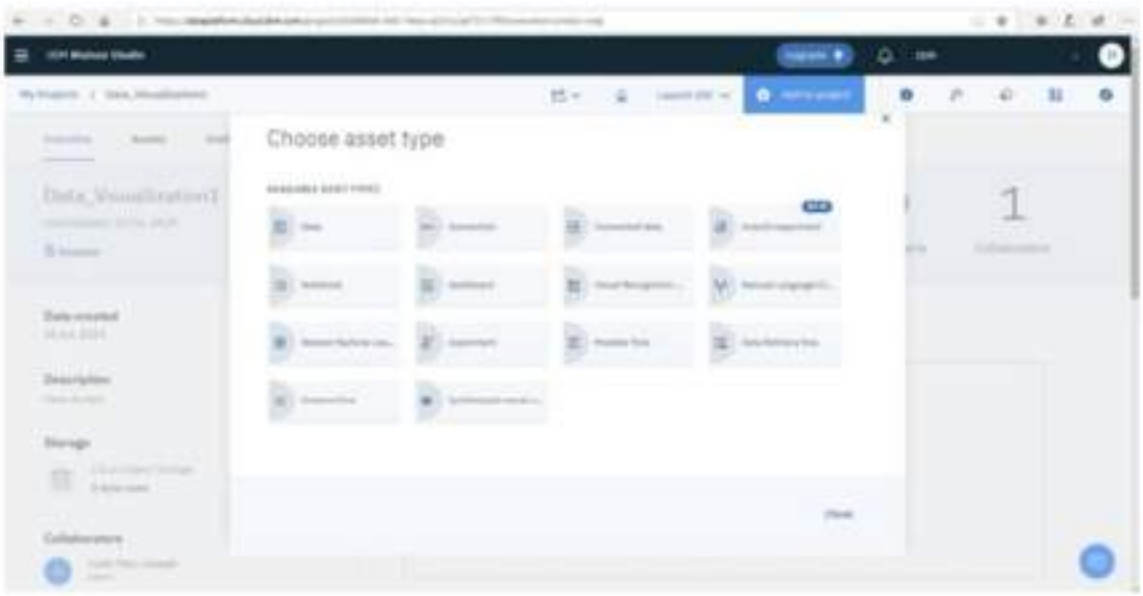
Click Create. You can start adding resources if your project is empty or begin working with the resources you imported.



To add data files to a project:

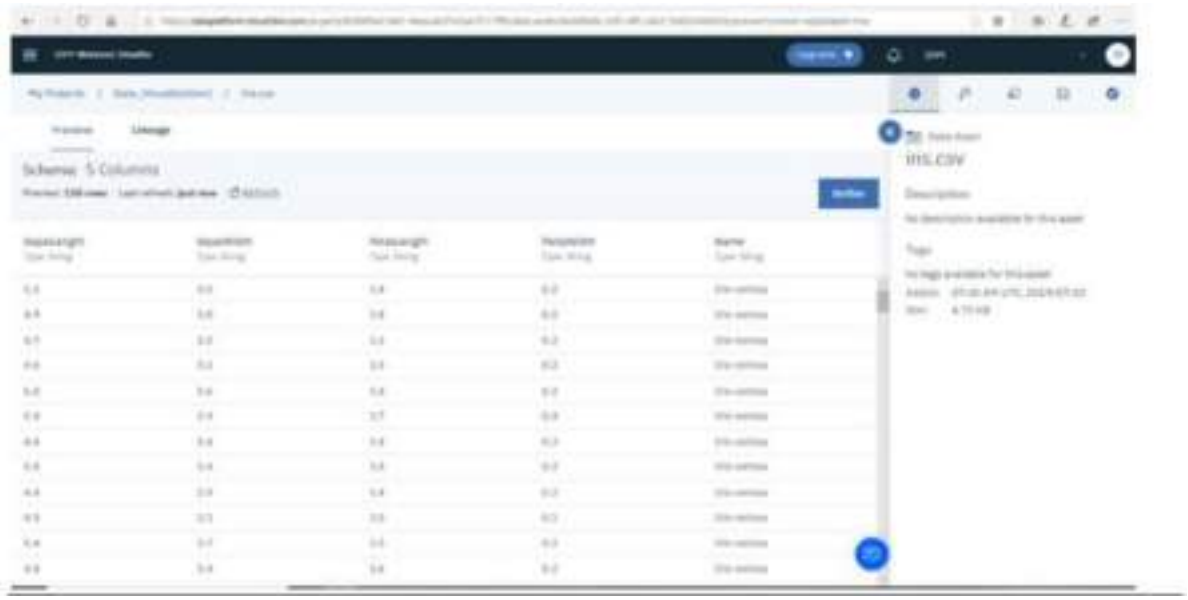
From your project's Assets page, click Add to project > Data or click the Find and add data icon ().You can also click the Find and add data icon from within a notebook or canvas.

In the Load pane that opens, browse for the files or drag them onto the pane. You must stay on the page until the load is complete. You can cancel an ongoing load process if you want to stop loading a file.



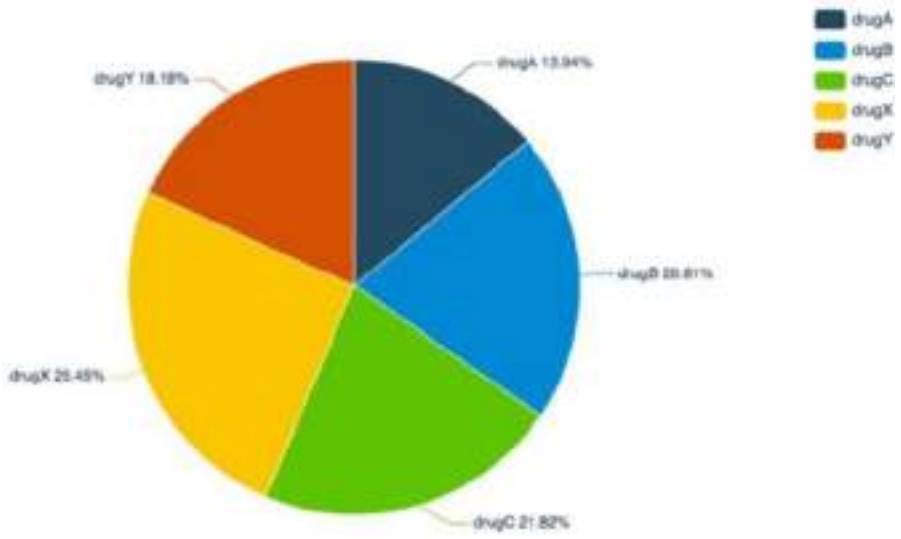
Case Study:

Let us take the Iris Data set to see how we can visualize the data in Watson studio.



Adding Data to Data Refinery

Visualizing information in graphical ways can give you insights into your data. By enabling you to look at and explore data from different perspectives, visualizations can help you identify patterns, connections, and relationships within that data as well as understand large amounts of information very quickly. You can also visualize your data with these same charts in an SPSS Modeler flow. Right-click a node and select **Profile.**



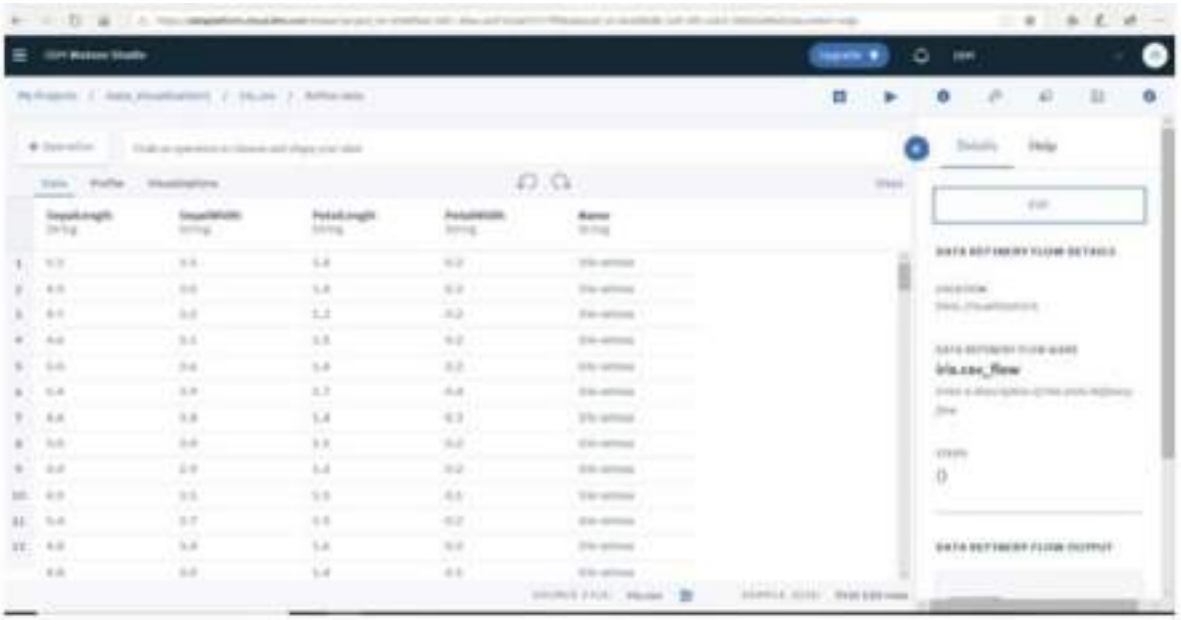
To visualize your data:

From Data Refinery, click the **Visualizations** tab.

Start with a chart or select columns.

1. Click any of the available charts. Then add columns in the **DETAILS** panel that opens on the left side of the page.
2. Select the columns that you want to work with. Suggested charts will be indicated with a dot next to the chart name. Click a chart to visualize your data.

Click on refine



Click on Visualization tab:

Add the columns by selecting.