Data Pre-processing I (Continuous and Categorical Data)

Applied Machine Learning for Educational Data Science

true

08/23/2021

Contents

[Updated: Fri, Oct 08, 2021 - 21:09:15]

1. Scales of Measurement and Types of Variables

It is important to understand the nature of variables and how they were measured and represented in a dataset. In social sciences, in particular psychology, there is a methodological consensus about the framework provided by Stevens (1946), also see Michell (2002) for an in-depth discussion. According to Stevens' definition, there are four levels of measurement: nominal, ordinal, interval, and ratio. Whether a variable is considered having a nominal, ordinal, interval, or ratio scale depends on the character of the empirical operations performed while constructing the variable.

- Nominal scale: Variables with a nominal scale cannot be be meaningfully added, subtracted, divided, or multiplied. Also, there is no hierarchical order among the assigned values. Most variables that contains labels for individual observations can be considered as nominal, e.g., hair color, city, state, ethnicity.
- Ordinal scale: Variables with an ordinal scale also represent labels; however, there is a meaningful hierarchy among the assigned values. For instance, if a variable is coded as Low, Medium, and High, they are simply labels but we know that High represents something more than Medium, and Medium represents something higher than Low (High > Medium > Low). On the other side, the distance between the assigned values do not necessarily represent the same amount of difference. Other examples of variables that can be considered as ordinal are letter grades (A-F), scores from likert type items (Strongly agree, agree, disagree, strongly disagree), education status(high school, college, master's, PhD), cancer stage (stage1, stage2, stage3), order of finish in a competition (1st, 2nd, 3rd).
- Interval scale: Variables with an ordinal scale represents quantities with equal measurement units but they don't have an absolute zero point. For instance, a typical example of an interval scale is temperature measured on the Fahrenheit scale. The difference between 20F and 30F is the same difference as the difference between 60F and 70F. However, 0F does not indicate no heat.
- Ratio scale: Variables with a ratio scale represents quantities with equal measurement units and have an absolute zero. Due to the nature of the existence of absolute zero point that represent 'nothing', ratio of measurements are also meaningful. Typical examples are height, mass, distance, length.

Below table provides a summary of properties for each scale.

	Indicating Difference	Indicating Direction of Difference	Indicating Amount of Difference	Has absolute zero
Nominal	X			
Ordinal	X	X		
Interval	X	X	X	
Ratio	X	X	X	X

In this class, we classify the variables in two types: **Categorical** and **Continuous**. The variables with a nominal or ordinal scale are considered as **Categorical** and the variables with an interval or ratio scale are considered as **Continuous**.

NOTE

In the following sections, you will see some R code to process different types of variables using the base R functions to introduce certain types of coding schemes. At the end, I will introduce the `recipes` package to provide a more complete picture of how to do all these different types of encoding in a more efficient manner for this specific dataset.

---

2. Processing Categorical Variables

When there are categorical predictors in a dataset, it is important to translate them into numerical codes. When encoding categorical predictors, we try to preserve as much information as possible from its labels. Therefore, different strategies may be used for categorical variables with different ordinal scales.

2.1 One-hot encoding (Dummy Variables)

A dummy variable is a synthetic variable with two outcomes (0 and 1) to represent a group membership. When there is a nominal variable with N levels, it is typical to create N dummy variables to represent the information in the nominal variable. Each dummy variable represents a membership to one of the levels in the nominal variable. These dummy variables can be used as features in predictive models.

In its simplest case, consider variable `Race` in the Recidivism dataset with two levels: Black and White. We can create two dummy variables such that the first dummy variable represents whether or not an individual is Black and the second dummy variable represents whether or not the individual is White.

	Dummy Variable 1	Dummy Variable 2
Black	1	0
White	0	1

```
recidivism <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/mai

table(recidivism$Race)
```

```
BLACK WHITE
10222  7889
```

```
recidivism$black <- ifelse(recidivism$Race=='BLACK',1,0)
recidivism$white <- ifelse(recidivism$Race=='WHITE',1,0)

head(recidivism[,c('Race','black','white')])
```

```
   Race black white
1 BLACK     1     0
2 BLACK     1     0
3 BLACK     1     0
4 WHITE     0     1
5 WHITE     0     1
6 BLACK     1     0
```

```
table(recidivism$black)
```

```
    0     1
 7889 10222
```

```
table(recidivism$white)
```

```
    0     1
10222  7889
```

Let's consider another example from the Recidivism dataset. Variable `Prison_Offense` has five categories: Violent/Sex, Violent/Non-Sex, Property, Drug, Other. We can create five dummy variables using the following coding scheme.

	Dummy Variable 1	Dummy Variable 2	Dummy Variable 3	Dummy Variable 4	Dummy Variable 5
Violent/Sex	1	0	0	0	0
Violent/Non-Sex	0	1	0	0	0
Property	0	0	1	0	0
Drug	0	0	0	1	0
Other	0	0	0	0	1

Note that `Prison_Offence` is missing for a number of observations. You can fill-in the missing values prior to creating dummy variables using one of the methods we will discuss later. Alternatively, we can define Missing as the sixth category to preserve that information.

	Dummy Variable 1	Dummy Variable 2	Dummy Variable 3	Dummy Variable 4	Dummy Variable 5	Dummy Variable 6
Violent/Sex	1	0	0	0	0	0
Violent/Non-Sex	0	1	0	0	0	0
Property	0	0	1	0	0	0
Drug	0	0	0	1	0	0
Other	0	0	0	0	1	0
Missing	0	0	0	0	0	1

```
table(recidivism$Prison_Offense)
```

```
          Drug          Other       Property Violent/Non-Sex     Violent/Sex
          3852           1895           5330           4000            743
```

```
names(table(recidivism$Prison_Offense))
```

```
[1] "Drug"            "Other"           "Property"        "Violent/Non-Sex"
[5] "Violent/Sex"
```

```
recidivism$off_viosex   <- ifelse(recidivism$Prison_Offense=='Violent/Sex',1,0)
recidivism$off_vionosex <- ifelse(recidivism$Prison_Offense=='Violent/Non-Sex',1,0)
recidivism$off_property <- ifelse(recidivism$Prison_Offense=='Property',1,0)
recidivism$off_drug     <- ifelse(recidivism$Prison_Offense=='Drug',1,0)
recidivism$off_other    <- ifelse(recidivism$Prison_Offense=='Other',1,0)
recidivism$off_missing  <- ifelse(recidivism$Prison_Offense=='',1,0)

head(recidivism[,c('Prison_Offense','off_viosex','off_vionosex','off_property','off_drug','off_other','
```

```
    Prison_Offense off_viosex off_vionosex off_property off_drug off_other
1             Drug          0            0            0        1         0
2  Violent/Non-Sex          0            1            0        0         0
3             Drug          0            0            0        1         0
4         Property          0            0            1        0         0
5         Property          0            0            1        0         0
6             <NA>         NA           NA           NA       NA        NA
```

```
7             Drug          0             0             0        1         0
8  Violent/Non-Sex          0             1             0        0         0
9         Property          0             0             1        0         0
10 Violent/Non-Sex          0             1             0        0         0
   off_missing
1            0
2            0
3            0
4            0
5            0
6           NA
7            0
8            0
9            0
10           0
```

In some cases, when you have geographical location with a reasonable number of categories (e.g., counties or cities in a state, schools in a district), you can also create dummy variables to represent this information. In our case, the Recidivism dataset has a variable called `Residence_PUMA` indicating Public Use Microdata Area (PUMA) for the residence address at the time individual was released. There is a total of 25 unique codes (1-25) for this variable; however, these numbers are just labels. So, one can create 25 different dummy variables to represent 25 different PUMAs.

NOTE

When you fit a typical regression model without regularizaton using ordinary least-squares (OLS), a typical practice is to drop a dummy variable for one of the levels. So, for instance, if there are N levels for a nominal variable, you only have to create *(N-1)* dummy variables, as the Nth one has redundant information. The information regarding to the excluded category is represented in the intercept term. It creates a problem when you put all N dummy variables into the model, because the OLS procedure tries to invert a singular matrix and you will likely get an error message.

On the other hand, this is not an issue when you fit a regularized regression model, which will be the case in this class. Therefore, you do not need to drop one of the dummy variables and can include all of them in the analysis. In fact, it may be beneficial to keep the dummy variables for all categories in the model when regularization is used in regression. Otherwise, the model may produce different predictions depending on which category is excluded.

2.2. Label encoding

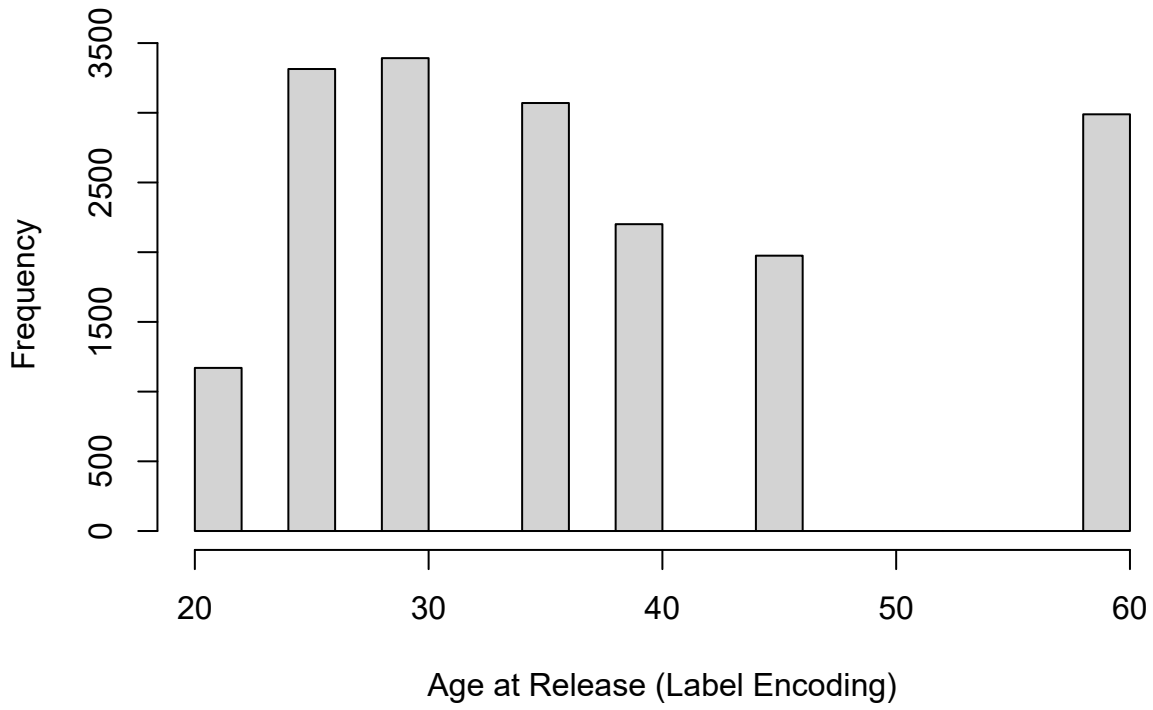
When the variable of interest is ordinal and there is a hierarchy among the levels, we can still use one-hot encoding to create a set of dummy variables to represent the information in the ordinal variable. However, dummy variables will not provide any information regarding the hierarchy among categories.

For instance, consider the variable `Age_At_Release` in the Recidivism dataset. It is coded as 7 different age intervals in the dataset: 18-22, 23-27, 28-32, 33-37, 38-42, 43-47, 48 or older. One can create 7 dummy variables to represent each category in this variable. Alternatively, one can assign a numeric variable to each category that may represent the information in these categories. For instance, one can assign numbers from 1 to 7, respectively. Or, one can choose the midpoint of each interval to represent each category (e.g., 20,25,31,35,40,45,60).

```
require(dplyr)
table(recidivism$Age_at_Release)
```

```
      18-22       23-27       28-32       33-37       38-42       43-47
       1170        3314        3392        3070        2201        1975
48 or older
       2989
```

```
#?dplyr::recode
recidivism$age <- recode(recidivism$Age_at_Release,
                         '18-22' = 20,
                         '23-27' = 25,
                         '28-32' = 30,
                         '33-37' = 35,
                         '38-42' = 40,
                         '43-47' = 45,
                         '48 or older' = 60)

hist(recidivism$age,main='',xlab='Age at Release (Label Encoding)')
```



Another example would be the variable `Education Level`. It has three different levels: At least some college, High School Diploma, Less than HS diploma. One can create three dummy variables to represent each level. Alternatively, one can assign 1, 2, and 3, respectively. Or, one can assign a number for the approximate years of schooling for each level such as 9, 12, and 15.

```
table(recidivism$Education_Level)

recidivism$edu <- recode(recidivism$Education_Level,
                         'At least some college' = 14,
                         'High School Diploma'   = 12,
                         'Less than HS diploma'  = 8)

hist(recidivism$edu,main='',xlab='Education Level (Label Encoding)')
```

2.3. Polynomial Contrasts

Another way of encoding an ordinal variable is to use polynomial contrasts. The polynomial contrasts may be helpful if one wants to explore whether or not there is a linear, quadratic, cubic, etc. relationship between the predictor variable and outcome variable. You can use `stat::poly()` function in R to obtain the set of polynomial contrasts. If there are *N* levels in an ordinal variable, then you can get polynomials up to degree *N-1*.

For instance, suppose you have an ordinal variable with three levels: Low, Medium, and High. Then, `stat::poly(x=1:3,degree=2)` will return the polynomial contrasts for the linear and quadratic terms. Notice that the input values for the `poly` function is a vector of numeric values that correspond to the levels in the ordinal variable, and the degree of polynomial terms requested. For this examle, it creates two sets of vectors to represent this ordinal variable. Note that the sum of the squares within each column is equal to 1, and the dot product of the contrast vectors is equal to 0. In other words, the polynomial terms represent a set of **orthonormal vectors**.

	Linear	Quadratic
Low	-0.707	0.408
Medium	0	-0.816
High	0.707	0.408

```
ctr <- poly(1:3,2)
ctr
```

```
                              1          2
[1,] -0.70710678118654757273731  0.4082483
[2,] -0.00000000000000007850462 -0.8164966
[3,]  0.70710678118654746171501  0.4082483
attr(,"coefs")
attr(,"coefs")$alpha
[1] 2 2

attr(,"coefs")$norm2
[1] 1.0000000 3.0000000 2.0000000 0.6666667

attr(,"degree")
[1] 1 2
attr(,"class")
[1] "poly"   "matrix"
```

```
sum(ctr[,1]^2)
```

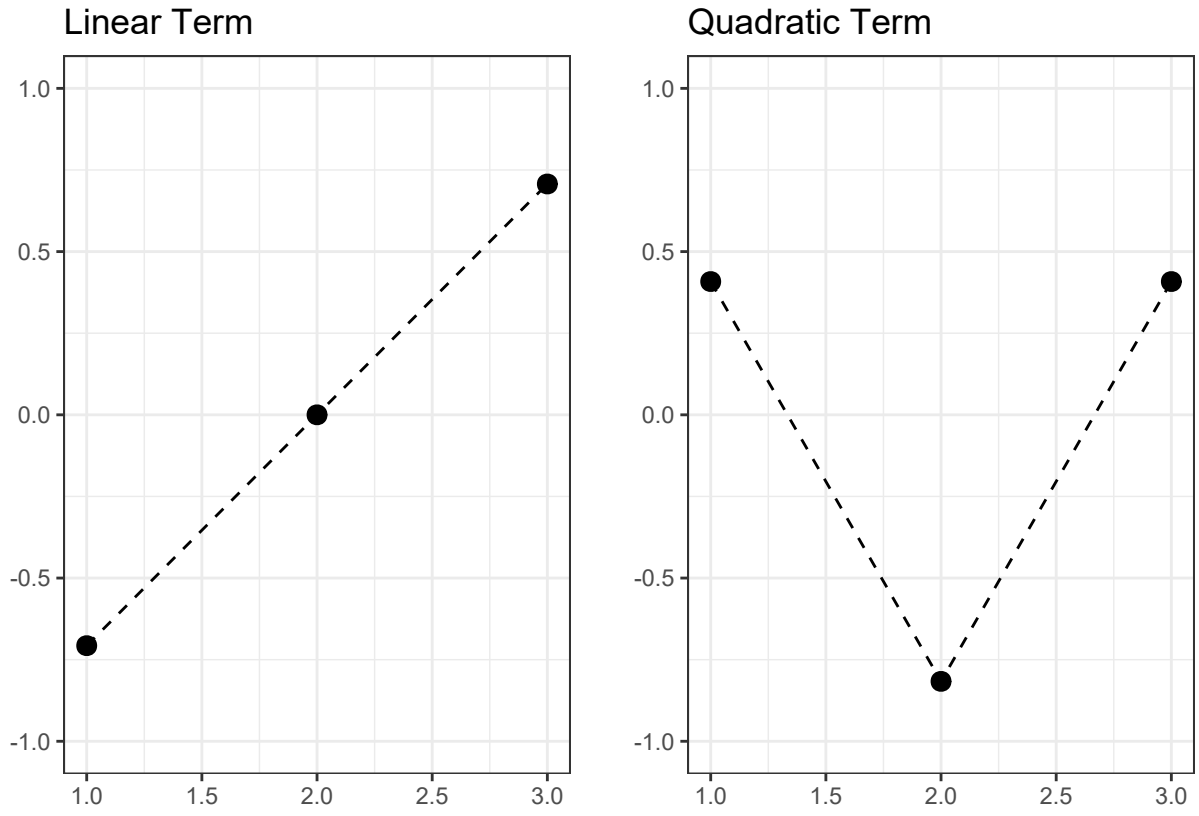
```
[1] 1
```

```
sum(ctr[,2]^2)
```

```
[1] 1
```

```
sum(ctr[,1]*ctr[,2])
```

```
[1] 0.00000000000000006410345
```

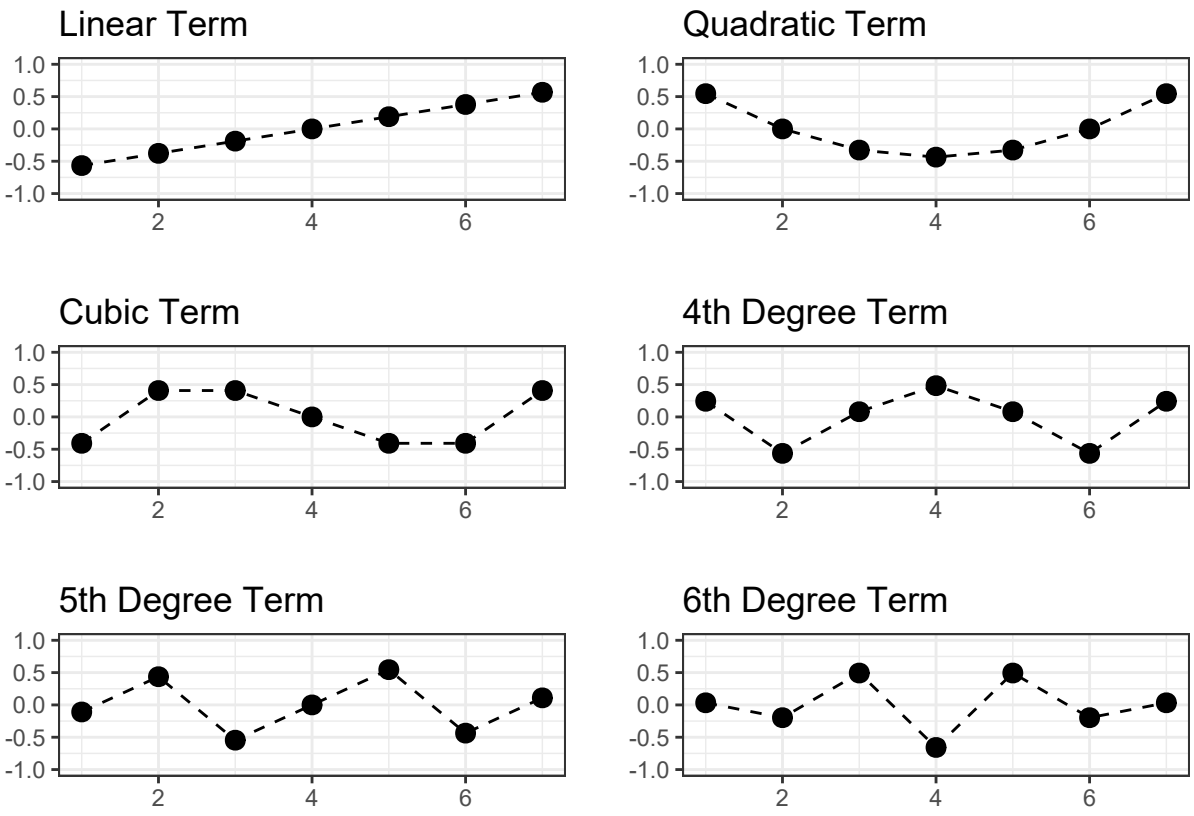


If we consider the variable `Age_At_Release` with 7 different levels, then we can have polynomial terms up to the 6th degree.

```
ctr <- poly(1:7,6)
ctr
```

```
                              1                           2
[1,] -0.566946709513840851890062  0.5455447255899807945667 1
[2,] -0.377964473009227308608 62  0.0000000000000000969082 1
[3,] -0.188982236504613571037 58 -0.3273268353539883768199 6
[4,]  0.000000000000000020981 24 -0.4364357804719848910046 7
[5,]  0.188982236504613571037 58 -0.3273268353539886543757 1
[6,]  0.377964473009227197586 31  0.0000000000000000000000 0
```

```
[7,]  0.5669467095138408518906  0.5455447255899810166113
                                3          4                          5
[1,] -0.4082482904638631282523  0.2417469 -0.1091089451179962505067
[2,]  0.4082482904638631837634 -0.5640761  0.4364357804719850575381
[3,]  0.4082482904638631282523  0.0805823 -0.5455447255899807945667
[4,]  0.0000000000000000302164  0.4834938 -0.0000000000000009751389
[5,] -0.4082482904638626841631  0.0805823  0.5455447255899812386559
[6,] -0.4082482904638630172300 -0.5640761 -0.4364357804719841138485
[7,]  0.4082482904638627951854  0.2417469  0.1091089451179959313176
              6
[1,]  0.03289758
[2,] -0.19738551
[3,]  0.49346377
[4,] -0.65795169
[5,]  0.49346377
[6,] -0.19738551
[7,]  0.03289758
attr(,"coefs")
attr(,"coefs")$alpha
[1] 4 4 4 4 4 4

attr(,"coefs")$norm2
[1]   1.0000   7.0000  28.0000  84.0000 216.0000 452.5714 685.7143 561.0390

attr(,"degree")
[1] 1 2 3 4 5 6
attr(,"class")
[1] "poly"   "matrix"
```



When we apply this to a variable in the dataset, these columns are rescaled such that they are orthonormal vectors. See below for an example:

```
# First, recode the Age_at_Release so that its levels are represented by numbers
# poly() doesn't work with character vectors

recidivism$age <- dplyr::recode(recidivism$Age_at_Release,
                                '18-22' = 1,
                                '23-27' = 2,
                                '28-32' = 3,
                                '33-37' = 4,
                                '38-42' = 5,
                                '43-47' = 6,
                                '48 or older' = 7)

# Create polynomial terms

poly.mat <- poly(x = recidivism$age, degree=6)
head(poly.mat)
```

```
                1           2           3           4           5
[1,]  0.0075566965 -0.001134569 -0.0094993975 -0.01283641494 -0.010472332
[2,] -0.0003464813 -0.008158592 -0.0005626538  0.00844393307  0.001808606
[3,]  0.0115082853  0.009369599  0.0064071011  0.00381687318  0.001776355
[4,]  0.0036051076 -0.006977299 -0.0089263309 -0.00006546544  0.010800278
```

```
[5,]  0.0036051076 -0.006977299 -0.0089263309 -0.00006546544  0.010800278
[6,]  0.0115082853  0.009369599  0.0064071011  0.00381687318  0.001776355
                 6
[1,] -0.0052954033
[2,] -0.0113555065
[3,]  0.0005831617
[4,]  0.0118791703
[5,]  0.0118791703
[6,]  0.0005831617
```

```
# Append them to the original data

recidivism$age_poly1 <- poly.mat[,1]
recidivism$age_poly2 <- poly.mat[,2]
recidivism$age_poly3 <- poly.mat[,3]
recidivism$age_poly4 <- poly.mat[,4]
recidivism$age_poly5 <- poly.mat[,5]
recidivism$age_poly6 <- poly.mat[,6]

head(recidivism[,c('Age_at_Release','age','age_poly1','age_poly2',
                   'age_poly3','age_poly4','age_poly5','age_poly6')],10)
```

```
   Age_at_Release age     age_poly1    age_poly2     age_poly3      age_poly4
1           43-47   6  0.0075566965 -0.001134569 -0.0094993975 -0.01283641494
2           33-37   4 -0.0003464813 -0.008158592 -0.0005626538  0.00844393307
3     48 or older   7  0.0115082853  0.009369599  0.0064071011  0.00381687318
4           38-42   5  0.0036051076 -0.006977299 -0.0089263309 -0.00006546544
5           38-42   5  0.0036051076 -0.006977299 -0.0089263309 -0.00006546544
6     48 or older   7  0.0115082853  0.009369599  0.0064071011  0.00381687318
7           38-42   5  0.0036051076 -0.006977299 -0.0089263309 -0.00006546544
8           43-47   6  0.0075566965 -0.001134569 -0.0094993975 -0.01283641494
9     48 or older   7  0.0115082853  0.009369599  0.0064071011  0.00381687318
10          33-37   4 -0.0003464813 -0.008158592 -0.0005626538  0.00844393307
      age_poly5     age_poly6
1  -0.010472332 -0.0052954033
2   0.001808606 -0.0113555065
3   0.001776355  0.0005831617
4   0.010800278  0.0118791703
5   0.010800278  0.0118791703
6   0.001776355  0.0005831617
7   0.010800278  0.0118791703
8  -0.010472332 -0.0052954033
9   0.001776355  0.0005831617
10  0.001808606 -0.0113555065
```

Notice that the values assigned to each category are different than the ones you see above, because they are rescaled such that polynomial terms have a unit length (sum of squared values equal to 1) and they are independent of each other (dot product is zero).

```
# Length of each polynomial term (sum of square values)

colSums(recidivism[,c('age_poly1','age_poly2','age_poly3','age_poly4','age_poly5','age_poly6')]^2)
```

```
age_poly1 age_poly2 age_poly3 age_poly4 age_poly5 age_poly6
        1         1         1         1         1         1
```

```
# The dot product is zero for each pair of 6 terms, see for a few examples

sum(recidivism[,c('age_poly1')]*recidivism[,c('age_poly2')])
```

```
[1] -0.00000000000004808991
```

```
sum(recidivism[,c('age_poly1')]*recidivism[,c('age_poly3')])
```

```
[1] -0.00000000000007869722
```

```
sum(recidivism[,c('age_poly4')]*recidivism[,c('age_poly6')])
```

```
[1] 0.0000000000000119
```

NOTE

There are other ways of encoding nominal and ordinal variables (e.g., Helmert contrasts), or one can come up with their own set of contrast values. When the goal of analysis is inference and you run analysis to respond to a specific research question, your research question typically dictates the type of encoding to use. You choose a coding scheme that provides you the most interpretable coefficients to respond to your research question.

On the other hand, when the goal of analysis is prediction, how you encode your categorical variable does not make much difference. In fact, they provide very similar predictions. Below, I provide an example using Age_at_Release variable to predict the outcome using different coding schemes and report the average squared error of predictions from a logistic regression model.

Encoding	Average Squared Error
Intercept-Only (NULL)	0.1885789
Dummy Variables	0.1861276
Label Encoding	0.1861888
Polynomial Contrasts	0.1861276
Helmert Contrasts	0.1861276

Notice that one-hot encoding, polynomial contrasts, and helmert contrasts have identical performance. In fact, they yield the exact same predicted value for observations. Moreover, a simple label encoding with a single constructed variable does (almost) as well as other encoding types with multiple constructed variables.

---

3. Processing Cyclic Variables

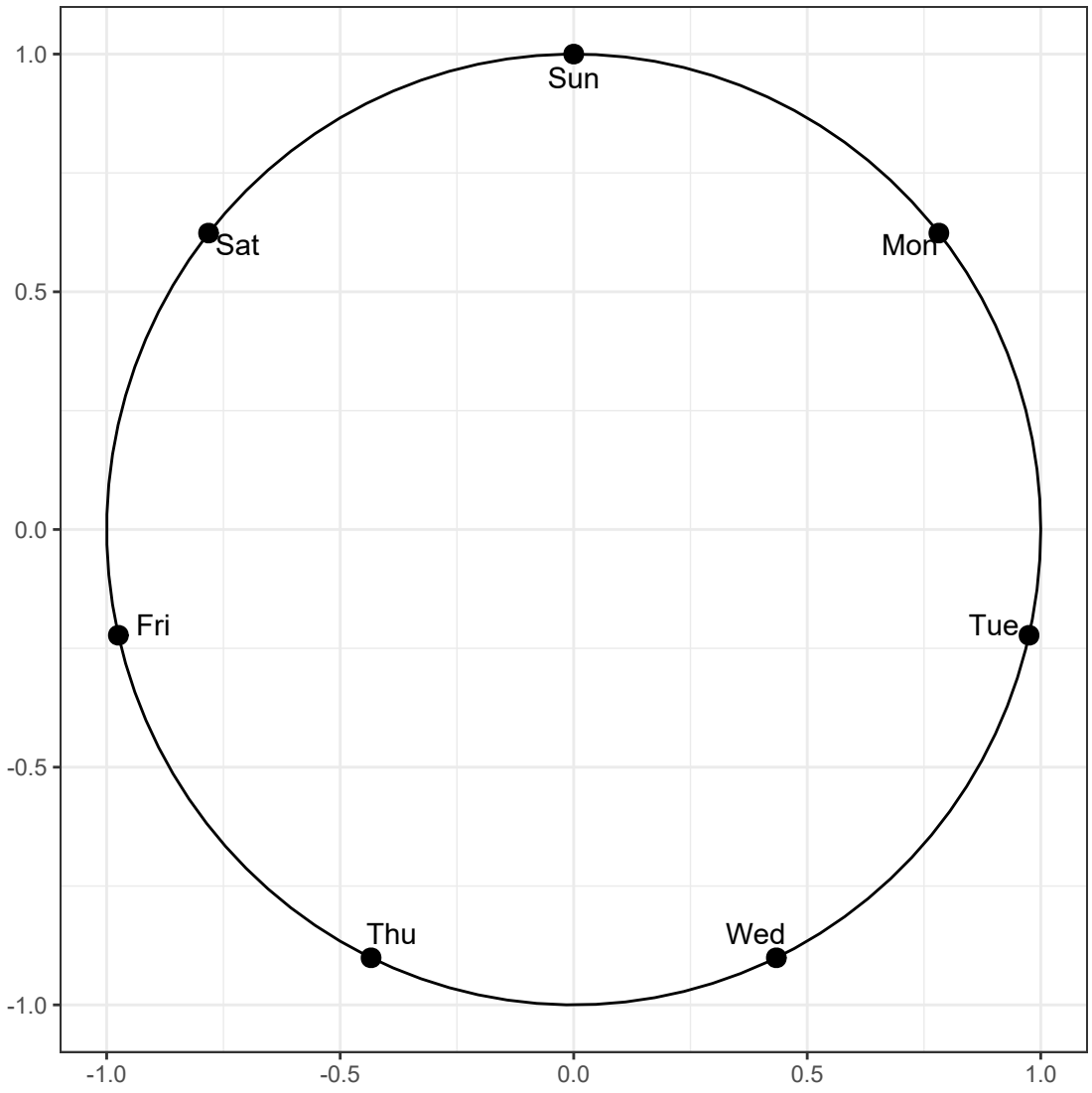
There are sometimes variables that are cyclic by nature (e.g., months, days, hour), and a type of encoding that represents their cyclic nature may be the most meaningful way to represent them instead of numerical or categorical encoding. One way to achieve this is to create two new variables using a sine and cosine transformation as the following:

$$x_1 = sin(\frac{2\pi x}{max(x)}),$$
$$x_2 = cos(\frac{2\pi x}{max(x)}).$$

For instance, suppose one of the variables in a dataset is the day of the week. We can represent its cyclic nature using the two variables as defined the following. Once the corresponding coordinates are calculated for each day of the week, the single variable that represents days in the data can be replaced with these two variables representing their coordinates in a unit circle.

```
d <- data.frame(days = c('Mon','Tue','Wed','Thu','Fri','Sat','Sun'),
                x = 1:7)

d$x1 <- sin((2*pi*d$x)/7)
d$x2 <- cos((2*pi*d$x)/7)

d
```

```
  days x                         x1         x2
1  Mon 1  0.781831482468029803634 1  0.6234898
2  Tue 2  0.974927912181823619342 0 -0.2225209
3  Wed 3  0.433883739117558231424 0 -0.9009689
4  Thu 4 -0.433883739117558009379 4 -0.9009689
5  Fri 5 -0.974927912181823619342 0 -0.2225209
6  Sat 6 -0.781831482468029914656 4  0.6234898
7  Sun 7 -0.000000000000000244921 3  1.0000000
```

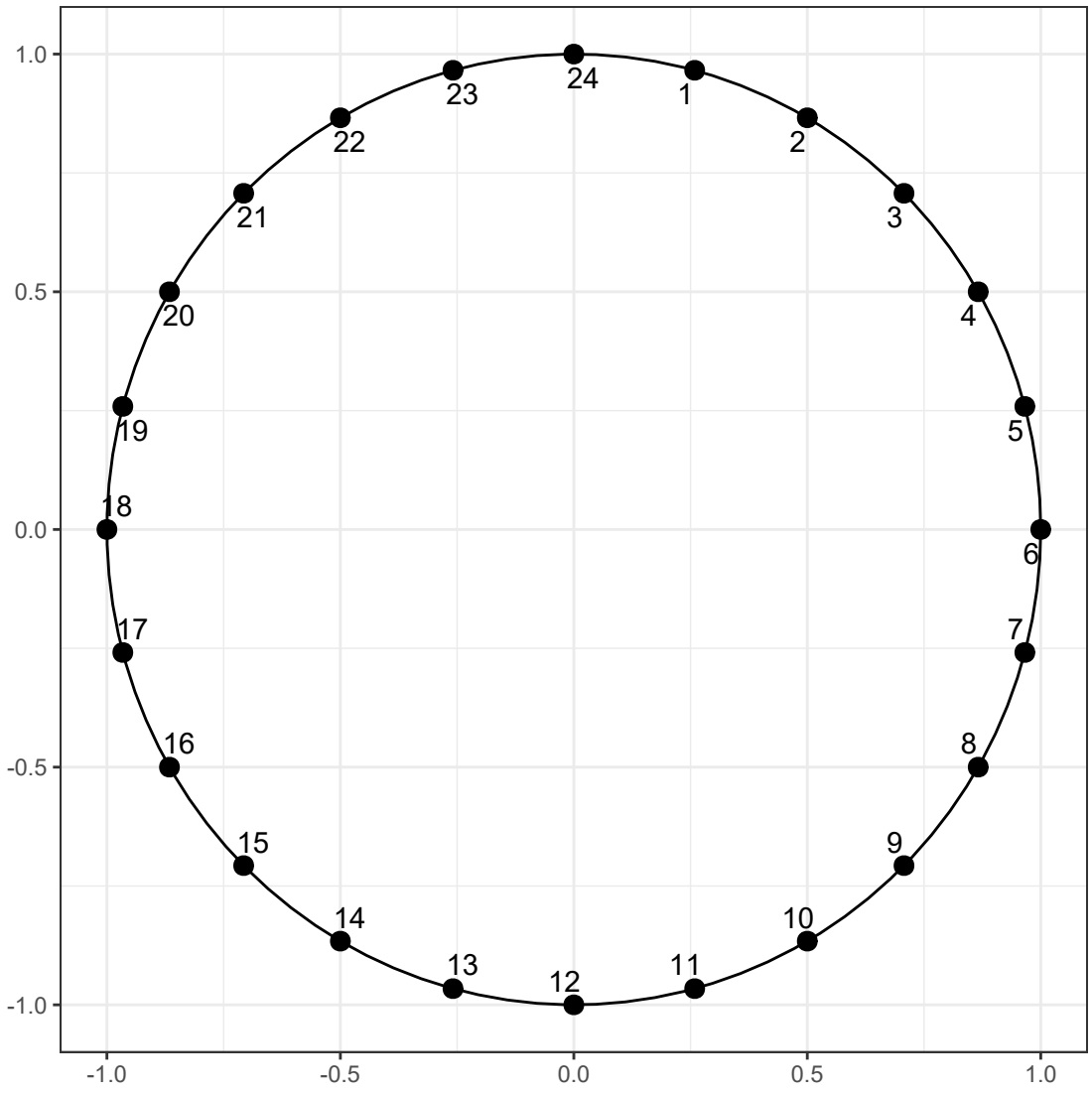


We can apply the same concept to any cyclic variable. Here is another example for the time of day.

```
d <- data.frame(hour = 1:24)

d$x1 <- sin((2*pi*d$hour)/24)
d$x2 <- cos((2*pi*d$hour)/24)

d
```

```
   hour                          x1                          x2
1     1  0.25881904510252073947640  0.96592582628906831221371
2     2  0.49999999999999944888  0.86602540378443870761060
3     3  0.70710678118654757273731  0.70710678118654757273731
4     4  0.86602540378443865965883  0.50000000000000011102230
5     5  0.96592582628906831221371  0.25881904510252073947640
6     6  1.00000000000000000000  0.00000000000000006123032
```

```
7     7  0.965925826289068312213 7 -0.258819045102520628454105
8     8  0.866025403784438707610 6 -0.499999999999997779554
9     9  0.707106781186547572737 3 -0.707106781186547461715
10   10  0.499999999999999444888 -0.866025403784438707610 6
11   11  0.258819045102521017032 2 -0.965925826289068201191 4
12   12  0.000000000000000122460 6 -1.000000000000000000000
13   13 -0.258819045102520794987 6 -0.965925826289068312213 7
14   14 -0.499999999999999722444 2 -0.866025403784438818632 9
15   15 -0.707106781186547128648 1 -0.707106781186547905804 2
16   16 -0.866025403784438374543 7 -0.500000000000000444089 21
17   17 -0.965925826289068312213 7 -0.258819045102520628454105
18   18 -1.000000000000000000000 0 -0.000000000000000183690 95
19   19 -0.965925826289068423236 0  0.258819045102520295387 19
20   20 -0.866025403784438596588 3  0.500000000000000111022 30
21   21 -0.707106781186547683759 6  0.707106781186547350692 71
22   22 -0.500000000000000444089 2  0.866025403784438374543 70
23   23 -0.258819045102521572143 7  0.965925826289068090169 11
24   24 -0.000000000000000244921 3  1.000000000000000000000 0
```



4. Processing Continuous Variables

4.1. Centering and Scaling (Standardization)

Centering a variable is done by substracting the mean of the variable from every value of the variable. This ensures that the mean of the centered variable is equal to zero. Scaling a variable is by dividing every value of variable by its standard deviation. When centering and scaling are both applied, this is called standardizing a variable. When we standardize a variable, we make sure that its mean is equal to zero and variance is equal to 1. Standardization may be important for certain types of models (e.g., K-nearest neighbor, support vector machines, penalized regression).

Below is an example from the Recidivism dataset using the variable `Avg_Days_per_DrugTest`.

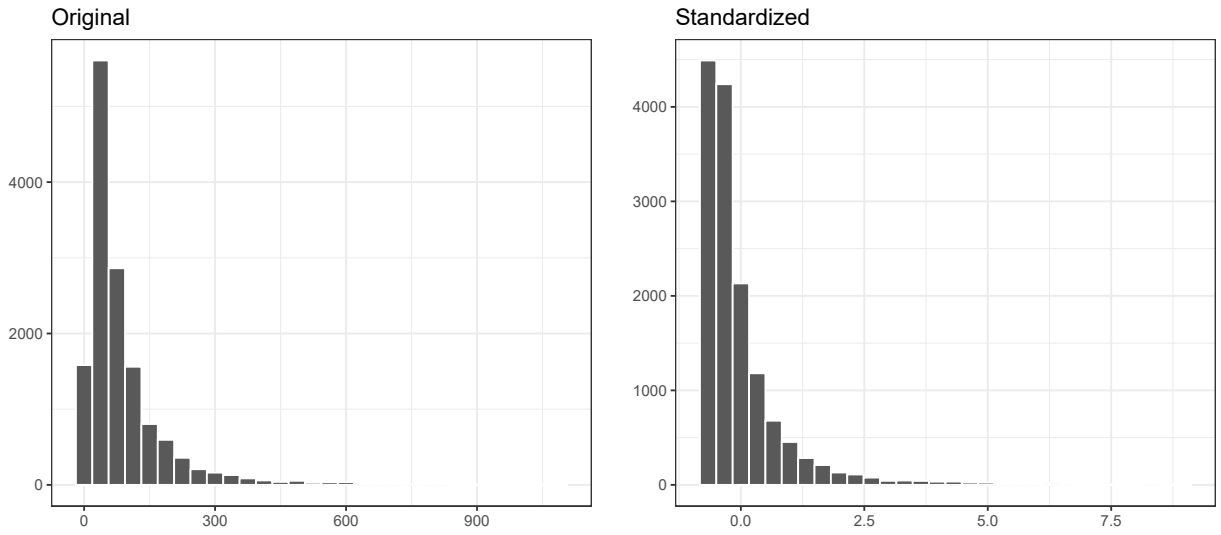
```
require(psych)
```

```r
# Descriptive stats for the original variable

describe(recidivism$Avg_Days_per_DrugTest)
```

```
   vars     n  mean     sd median trimmed   mad  min    max   range skew
X1    1 14343 92.41 113.11  56.03   69.29 47.28 1.27 1088.5 1087.23 3.64
   kurtosis   se
X1    18.24 0.94
```

```r
  # Center

  recidivism$Avg_Days_per_DrugTest2 <- recidivism$Avg_Days_per_DrugTest - mean(recidivism$Avg_Days_per_D

  # Scale

  recidivism$Avg_Days_per_DrugTest2 <- recidivism$Avg_Days_per_DrugTest2/sd(recidivism$Avg_Days_per_Drug

# Descriptive stats for the standardized variable

describe(recidivism$Avg_Days_per_DrugTest2)
```

```
   vars     n mean sd median trimmed  mad   min  max range skew kurtosis   se
X1    1 14343    0  1  -0.32    -0.2 0.42 -0.81 8.81  9.61 3.64    18.24 0.01
```



NOTE

Standardizing a variable only changes the first and second moments of a distribution (mean and variance); however, it doesn't change the third and fourth moments of a distribution (skewness and kurtosis). Notice that the skewness and kurtosis for both the original and the standardized variable are 3.64 and 18.24, respectively. We only change the mean to zero and variance to one by standardizing a variable.

4.2. Box-Cox transformation

Variables with extreme skewness and kurtosis may deteriorate the model performance for certain types of models. Therefore, it may sometimes be useful to transform a variable with extreme skewness and kurtosis such that its distribution approximates to a normal distribution. Box-Cox transformation is a method to find an optimal parameter of λ to apply the following transformation:

$$y^{(\lambda)} = \begin{cases} \frac{y^{\lambda}-1}{\lambda} & ,\lambda \neq 0 \\ ln(y) & ,\lambda = 0 \end{cases}$$

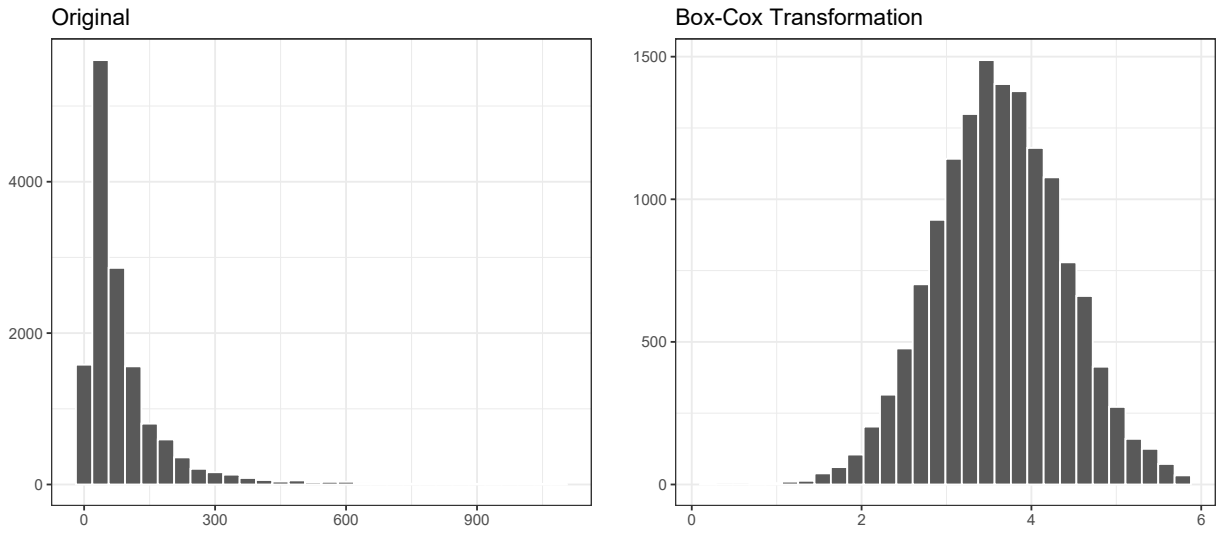
Below is an example of transforming the `Avg_Days_per_DrugTest` variable using the `boxcox` function from the `bestNormalize` package. Notice that the skewness and the kurtosis for the transformed variable are 0 and -0.02, respectively.

```
require(bestNormalize)

fit <- boxcox(recidivism$Avg_Days_per_DrugTest,standardize=FALSE)
fit
```

```
Non-Standardized Box Cox Transformation with 14343 nonmissing obs.:
 Estimated statistics:
 - lambda = -0.05390421
 - mean (before standardization) = 3.628338
 - sd (before standardization) = 0.758383
```

```
recidivism$Avg_Days_per_DrugTest_norm <- predict(fit)

describe(recidivism$Avg_Days_per_DrugTest_norm)
```

```
   vars     n mean   sd median trimmed  mad  min  max range skew kurtosis   se
X1    1 14343 3.63 0.76   3.62    3.63 0.77 0.24 5.83  5.59    0    -0.02 0.01
```



NOTE

Box-Cox transformation can be used only for variables with positive values. Therefore, it is a good idea to first implement the Box-Cox transformation, and then standardize a variable if both procedures will be applied to a variable. If there is a variable with negative values or a mix of both positive and negative values, the Yeo-Johnson transformation is available as an extension of the Box-Cox transformation. See this link for more information. The function `yeojohnson` is available in the `bestNormalizer` package to implement the Yeo-Johnson transformation (See, ?bestNormalizer::yeojohnson).

4.3. Logit Transformation

When a variable is a proportion bounded between 0 and 1, the logit transformation can be applied such that

$$\pi^* = ln(\frac{\pi}{1-\pi}),$$

where π represents a proportion. This may be particularly useful when your outcome variable is a proportion bounded between 0 and 1. When a linear model is used to model an outcome bounded between 0 and 1, the model predictions may exceed the reasonable range of values (predictions equal to less than zero or greater than one). Logit transformation scales variables such that the range of values becomes $-\infty$ and ∞ on the logit scale. One can build a model to predict logit (π^*) instead of proportion (π), and then ensure that the model predictions are bounded between 0 and 1 on the original proportion scale after a simple reverse operation for predicted values.

$$\pi = \frac{e^{\pi^*}}{1+e^{\pi^*}}$$

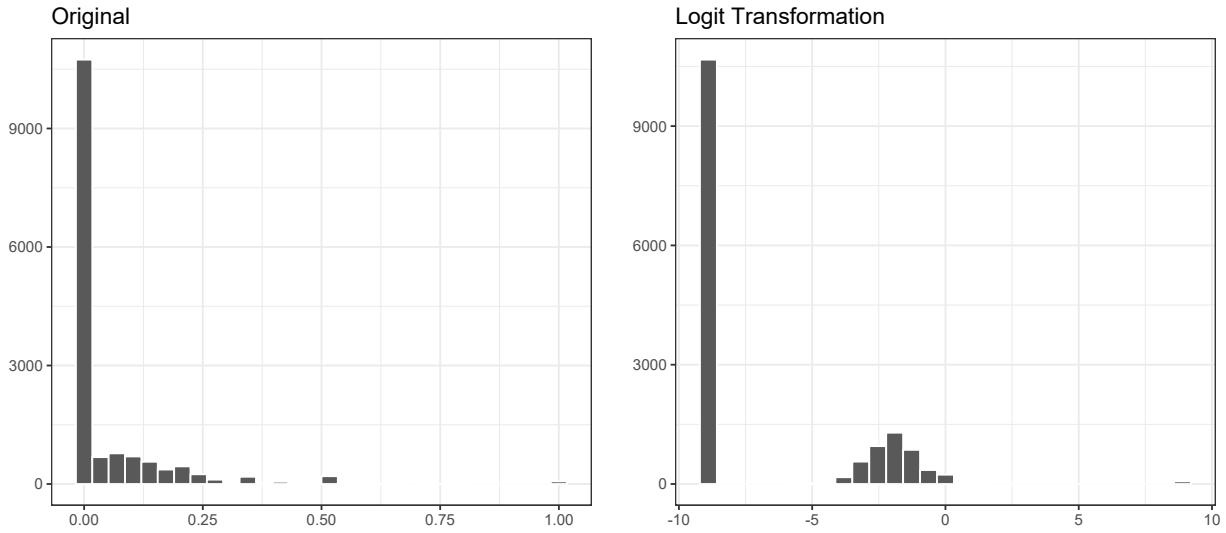
One caveat of using logit transformation is that it is not defined for 0 and 1. So, when you have values in the dataset exactly equal to 0 or 1, logit transformation will return either $-\infty$ and ∞. In these situations, we may add or substract a very tiny constant (e.g., .0001) to force the transformation to return a numerical value.

$$\pi^* = ln(\frac{\pi}{1-\pi}) = ln(\frac{0}{1-0}) = -\infty$$

$$\pi^* = ln(\frac{\pi}{1-\pi}) = ln(\frac{1}{1-1}) = \infty$$

Below is an example of logit transformation for the variable `DrugTests_THC_Positive` in the dataset. There is a large amount of zeros in this variable, and some ones. First, I change the values in the dataset such that all zeros become 0.0001 and all ones become 0.9999. Then, I apply the logit transformation. Notice that the distribution looks more symmetric except the extreme values.

```
recidivism$DrugTests_THC_Positive <- ifelse(recidivism$DrugTests_THC_Positive==0,
                                            .0001,
                                            recidivism$DrugTests_THC_Positive)

recidivism$DrugTests_THC_Positive <- ifelse(recidivism$DrugTests_THC_Positive==1,
                                            .9999,
                                            recidivism$DrugTests_THC_Positive)

recidivism$DrugTests_THC_Positive2 <- log(recidivism$DrugTests_THC_Positive/(1-recidivism$DrugTests_THC
```



4.4. Basis Expansions

Basis expansions are useful to address nonlinearity between a predictor variable and outcome variable. Using the basis expansions, one can create a set of feature variables using a nonlinear function of a variable x, $\phi(x)$. One simply replaces the original variable x with the new variables obtained from $\phi(x)$. For continuous predictors, the most commonly used expansions are polynomial basis expansions. The nth degree polynomial basis expansion can be represented by

$$\phi(x) = \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + ... + \beta_n x^n.$$

Suppose we have a variable x with values from 1 to 10. The third degree polynomial basis expansion (cubic basis expansion) can be found using the `poly` function as the following.

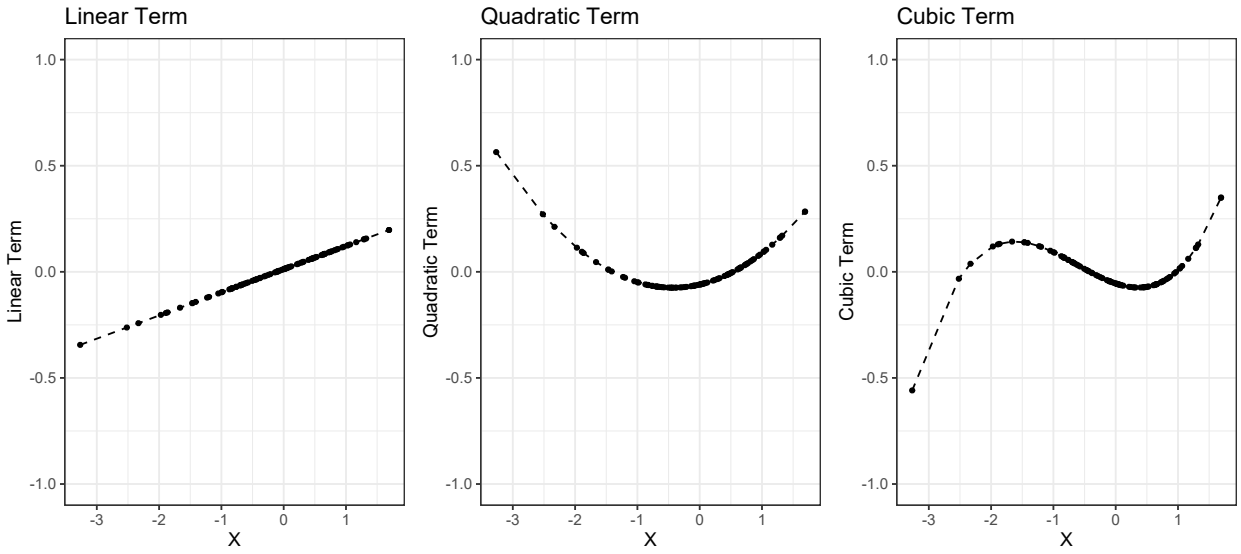
```
set.seed(654)

x <- rnorm(100,0,1)

head(x)
```

```
[1] -0.76031762 -0.38970450  1.68962523 -0.09423560  0.09530146  0.81727228
```

```
head(poly(x,degree=3))
```

```
                1           2           3
[1,] -0.070492258 -0.06612854  0.056003658
[2,] -0.030023304 -0.07454585 -0.003988336
[3,]  0.197028288  0.28324096  0.348896805
[4,]  0.002240307 -0.06560960 -0.044790680
[5,]  0.022936731 -0.05256865 -0.063289287
[6,]  0.101772051  0.04942613 -0.034439696
```

So, one can use these three new variables representing a linear, quadratic, and cubic trend in our prediction model instead of the original variable x. See below the relationship between the original variable x and the new polynomial features to replace it.



For continuous predictors, there is no limit for the degree of polynomial. The higher the degree of polynomial, the more flexible the model becomes, and there is a higher chance of overfitting. Typically, polynomial terms up to the 3rd or 4th degree are more than enough.

One drawback of the polynomial basis expansions is their non-locality. In other words, the value for a polynomial term at x_0 depends on data values far from x_0. This may lead unstable behavior at the edge of the distribution of variable x. Therefore, polynomial splines may provide a better representation. In polynomial splines, the range of variable x is divided into multiple regions and a separate polynomial piece is created within each region. The points that divide the regions are called *knots*. One particular class of basis functions that represent polynomial splines are called B-splines that can be and can be obtained using the `bs` function from the `splines` package. The number of new features to be created is equal to the sum of the number of knots and the degree of polynomial.

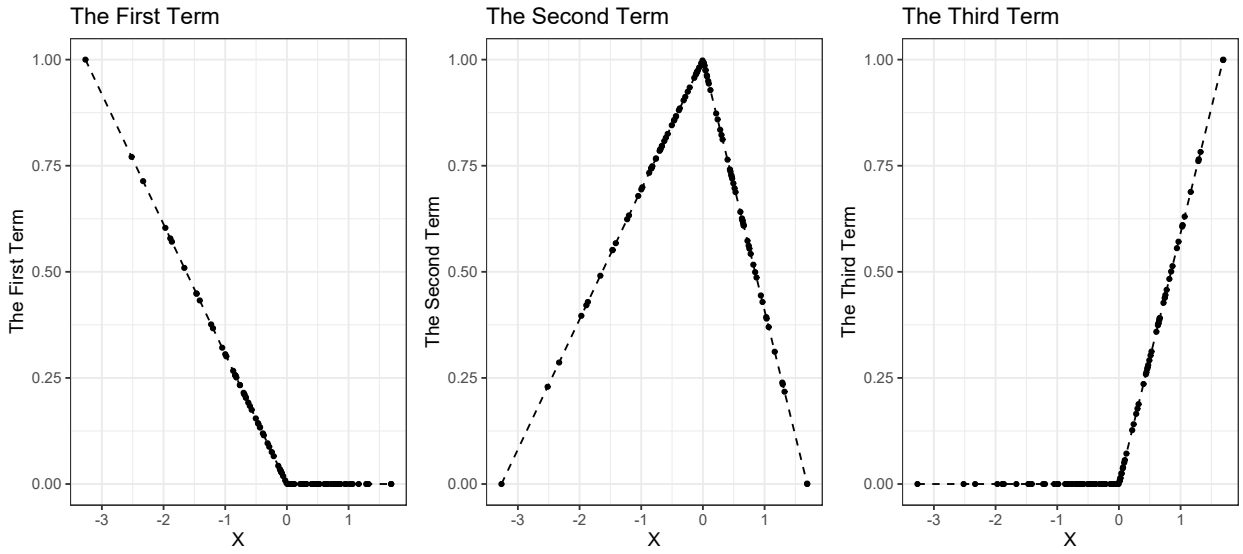
Suppose one wants to create B-splines with the first degree polynomial and a single knot (τ_1). The basis function can be written as

$$\phi(x) = \beta_0 + \beta_1 x + \beta_2 (x - \tau_1).$$

For instance, `bs(x,degree=1,knot=0,intercept=TRUE)` will return a matrix with three columns representing the basis expansions of a first degree polynomial spline with a single knot at 0, $\tau = 0$.

```
require(splines)

head(bs(x,degree=1,knot=0,intercept=TRUE))
```

```
             1           2          3
[1,] 0.23273681 0.767263190 0.00000000
[2,] 0.11929039 0.880709614 0.00000000
[3,] 0.00000000 0.001057462 0.99894254
[4,] 0.02884596 0.971154039 0.00000000
[5,] 0.00000000 0.943655741 0.05634426
[6,] 0.00000000 0.516811167 0.48318883
```



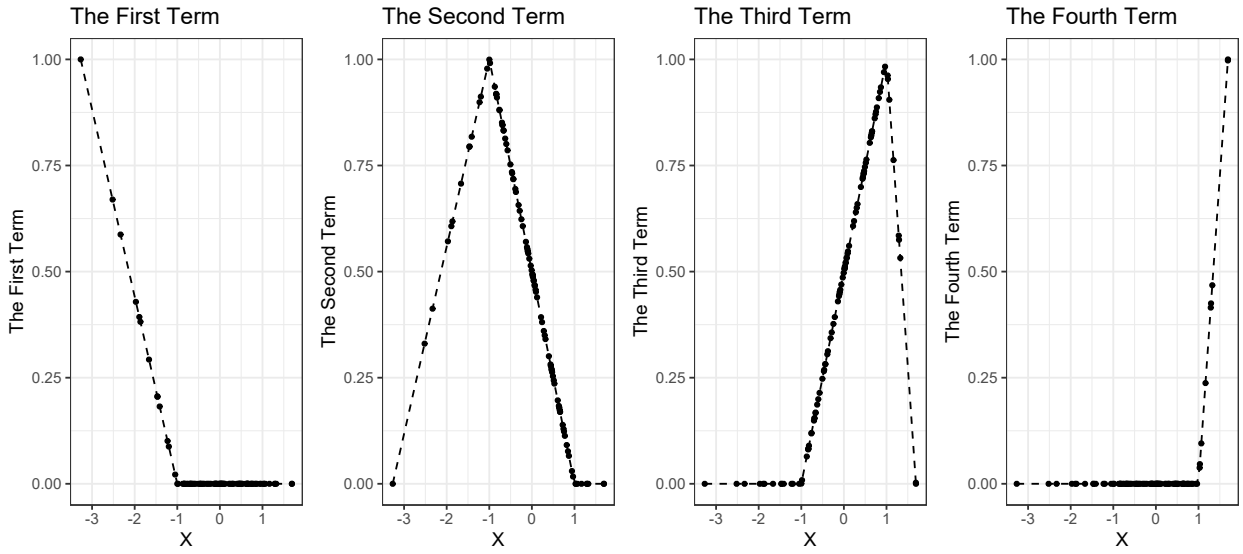
Instead of a single knot, one can request the basis expansions of a first degree polynomial spline with two knots, τ_1 and τ_2. Then the basis function can be written as

$$\phi(x) = \beta_0 + \beta_1 x + \beta_2(x - \tau_1) + \beta_3(x - \tau_2),$$

and `bs(x,degree=1,knot=c(-1,1),intercept=TRUE)` will return a matrix with four columns representing the basis expansions of a first degree polynomial spline with two knots at -1 and 1, $\tau_1 = -1$ and $\tau_2 = 1$.

```
head(bs(x,degree=1,knot=c(-1,1),intercept=TRUE))
```

```
     1          2           3         4
[1,] 0 0.88015881 0.119841191 0.0000000
[2,] 0 0.69485225 0.305147749 0.0000000
[3,] 0 0.00000000 0.002586882 0.9974131
[4,] 0 0.54711780 0.452882201 0.0000000
[5,] 0 0.45234927 0.547650730 0.0000000
[6,] 0 0.09136386 0.908636138 0.0000000
```



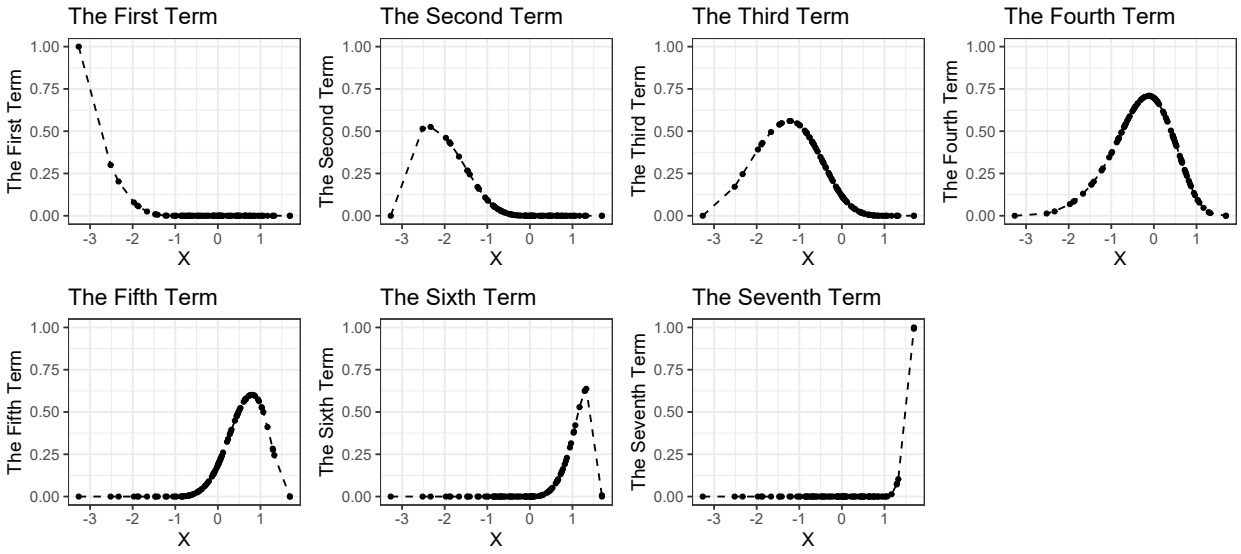
A cubic spline with three knots is typically enough for most practical applications. The basis function can be written as

$$\phi(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 + \beta_4 (x - \tau_1)^3 + \beta_5 (x - \tau_2)^3 + \beta_6 (x - \tau_3)^3,$$

and `bs(x,degree=3,knot=c(-1,0,1),intercept=TRUE)` will return a matrix with seven columns representing the basis expansions of cubic spline with three knots at -1, 0, and 1.

```
head(bs(x,degree=3,knot=c(-1,0,1),intercept=TRUE))
```

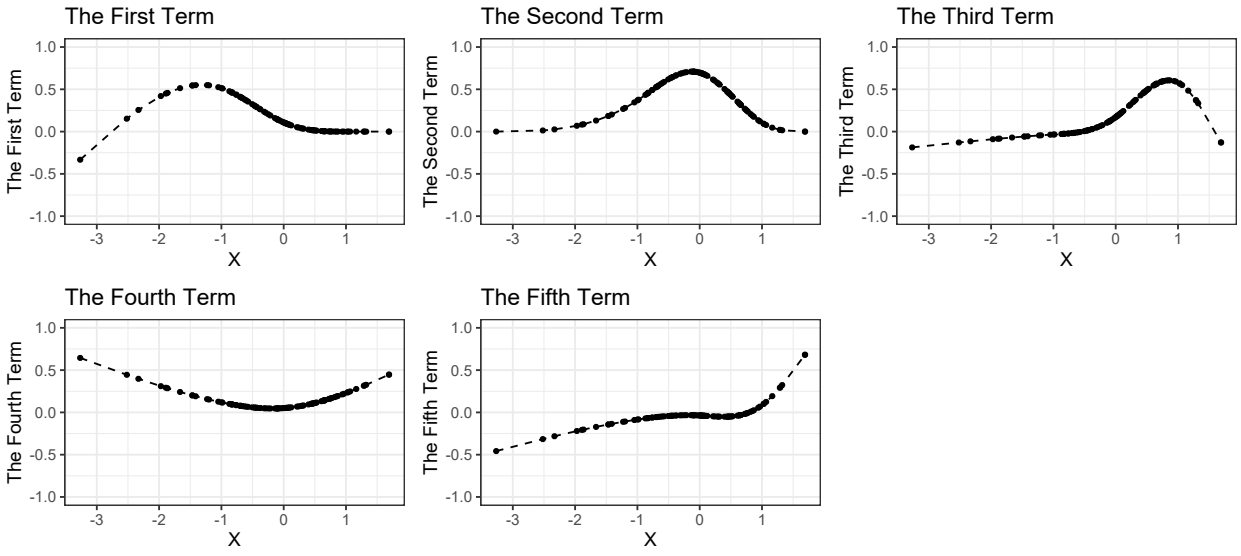
```
     1            2           3                 4              5            6
[1,] 0 0.04118368536 0.4634727495 0.492785580911832 0.002557984232 0.0000000000
[2,] 0 0.00554557155 0.2908441979 0.661381305706526 0.042228924817 0.0000000000
[3,] 0 0.00000000000 0.0000000000 0.000000001817923 0.000008194799 0.0077323894
[4,] 0 0.00007841245 0.1531958840 0.708676066749603 0.138049636782 0.0000000000
[5,] 0 0.00000000000 0.0867708193 0.669628136218787 0.243298493282 0.0003025512
[6,] 0 0.00000000000 0.0007149495 0.207088342367066 0.601386965719 0.1908097424
             7
[1,] 0.0000000
[2,] 0.0000000
[3,] 0.9922594
[4,] 0.0000000
[5,] 0.0000000
[6,] 0.0000000
```



Another class of basis functions that represent polynomial splines is Natural Splines. Natural splines are similar to B-splines with additional constraints that they are linear in the tails of the boundary knots, $(-\infty, \tau_1]$ and $(\tau_n, \infty]$. The natural splines may provide better behavior at the boundaries of the variable. Below is an example how to request natural cubic splines basis functions with three knots at -1, 0, and 1 using the `ns` function from the `splines` package.

```
head(ns(x,knot=c(-1,0,1),intercept=TRUE))
```

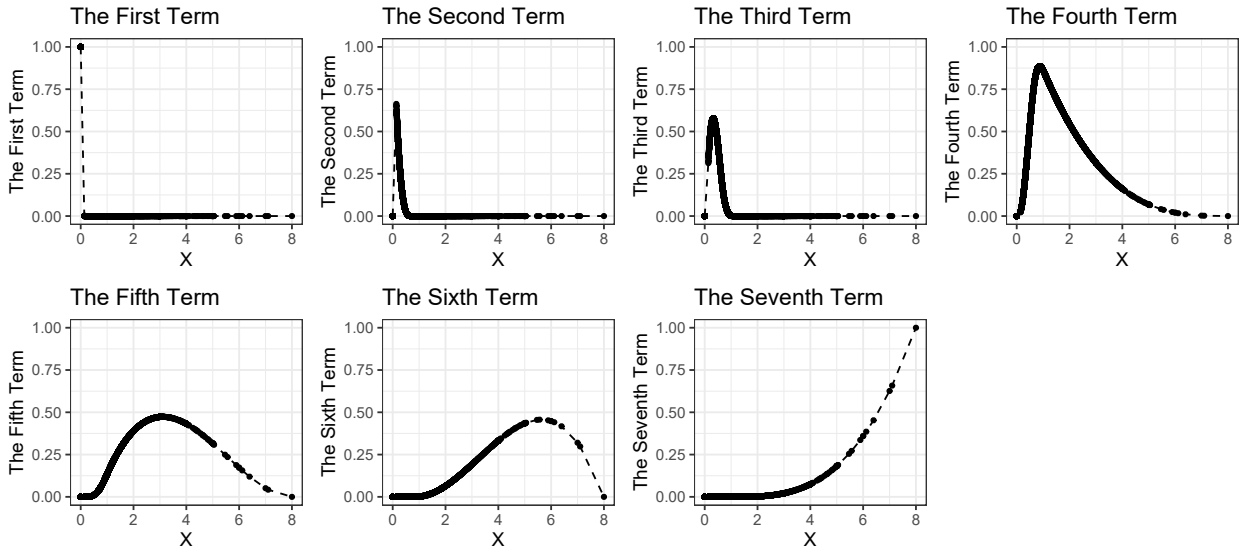
```
                1                2           3          4            5
[1,] 0.4363232496 0.492785580911832 -0.02232086 0.08574024 -0.060861403
[2,] 0.2701003089 0.661381305706526  0.02705323 0.05230017 -0.037124476
[3,] 0.0000000000 0.000000001817923 -0.12727410 0.44638688  0.680887225
[4,] 0.1417504054 0.708676066749603  0.12427909 0.04745759 -0.033687043
[5,] 0.0802799398 0.669628136218787  0.22680612 0.05714041 -0.040345487
[6,] 0.0006614678 0.207088342367066  0.60433667 0.18064414  0.007215901
```



Below, there is an example of cubic basis expansion of a continuous predictor `Jobs_Per_Year` using B-splines and Natural Splines at three knot points. I choose the knot locations such that they correspond to the 25th, 50th, and 75th percentile in the distribution of this variable.

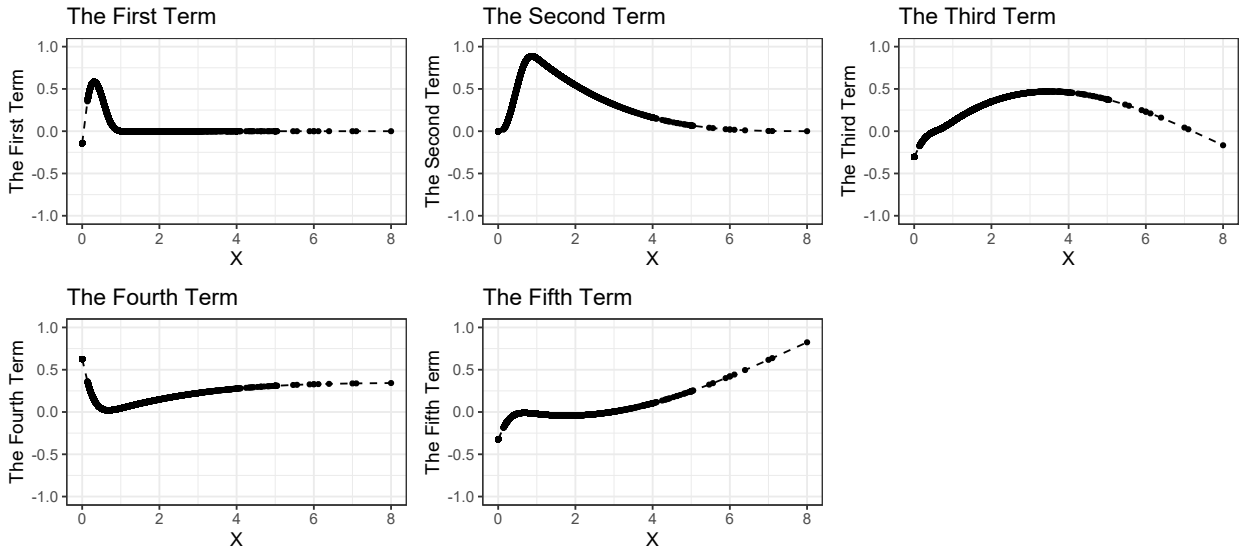
```
knots <- quantile(recidivism$Jobs_Per_Year,probs = c(.25,.5,.75),na.rm=TRUE)

head(bs(recidivism$Jobs_Per_Year,degree=3,knots = knots,intercept=TRUE))
```

```
     1          2         3         4            5             6           7
[1,] 0 0.05421945 0.4943019 0.4449983 0.0064803054 0.000000000000 0.000000000
[2,] 0 0.00000000 0.0000000 0.5440708 0.3895839604 0.063999439489 0.002345752
[3,] 1 0.00000000 0.0000000 0.0000000 0.0000000000 0.000000000000 0.000000000
[4,] 0 0.00000000 0.1198340 0.8336957 0.0464686200 0.000001714453 0.000000000
[5,] 0 0.21651919 0.5766527 0.2059080 0.0009200702 0.000000000000 0.000000000
[6,] 1 0.00000000 0.0000000 0.0000000 0.0000000000 0.000000000000 0.000000000
```




```
knots <- quantile(recidivism$Jobs_Per_Year,probs = c(.25,.5,.75),na.rm=TRUE)

head(ns(recidivism$Jobs_Per_Year,knots = knots,intercept=TRUE))
```

```
             1         2           3          4           5
[1,]  0.4918218 0.4449983 -0.02138791 0.05732613 -0.02945792
[2,]  0.0000000 0.5440708  0.34856595 0.14837528 -0.04101207
[3,] -0.1417101 0.0000000 -0.30449563 0.62636076 -0.32186513
[4,]  0.1183536 0.8336957  0.03596882 0.02160026 -0.01109874
[5,]  0.5840108 0.2059080 -0.06934574 0.14453983 -0.07427402
[6,] -0.1417101 0.0000000 -0.30449563 0.62636076 -0.32186513
```



5. Handling Missing Data

Missing data deserves a course of its own. For a comprehensive review of how to handle and impute missing data. For certain types of models such as gradient boosting, missing data is not a problem, and one can leave them as is without any processing. On the other hand, some models such as regularized regression models require complete data and one have to deal with missing data before modeling data.

First, we can do a very preliminary analysis about the amount of missing data. In the below code, we compute the proportion of missing values for 48 predictor variables and the outcome variable in the dataset using the `finalfit` package.

```
require(finalfit)

ff_glimpse(recidivism)$Continuous[,c('n','missing_percent')]
```

	n	missing_percent
ID	18111	0.0
Residence_PUMA	18111	0.0
Gang_Affiliated	15609	13.8
Supervision_Risk_Score_First	17819	1.6
Dependents	18111	0.0
Prior_Arrest_Episodes_Felony	18111	0.0
Prior_Arrest_Episodes_Misd	18111	0.0
Prior_Arrest_Episodes_Violent	18111	0.0
Prior_Arrest_Episodes_Property	18111	0.0
Prior_Arrest_Episodes_Drug	18111	0.0
Prior_Arrest_Episodes_PPViolationCharges	18111	0.0
Prior_Arrest_Episodes_DVCharges	18111	0.0
Prior_Arrest_Episodes_GunCharges	18111	0.0
Prior_Conviction_Episodes_Felony	18111	0.0
Prior_Conviction_Episodes_Misd	18111	0.0
Prior_Conviction_Episodes_Viol	18111	0.0
Prior_Conviction_Episodes_Prop	18111	0.0
Prior_Conviction_Episodes_Drug	18111	0.0

```
Prior_Conviction_Episodes_PPViolationCharges      18111             0.0
Prior_Conviction_Episodes_DomesticViolenceCharges 18111             0.0
Prior_Conviction_Episodes_GunCharges              18111             0.0
Prior_Revocations_Parole                          18111             0.0
Prior_Revocations_Probation                       18111             0.0
Condition_MH_SA                                   18111             0.0
Condition_Cog_Ed                                  18111             0.0
Condition_Other                                   18111             0.0
Violations_ElectronicMonitoring                   18111             0.0
Violations_Instruction                            18111             0.0
Violations_FailToReport                           18111             0.0
Violations_MoveWithoutPermission                  18111             0.0
Delinquency_Reports                               18111             0.0
Program_Attendances                               18111             0.0
Program_UnexcusedAbsences                         18111             0.0
Residence_Changes                                 18111             0.0
Avg_Days_per_DrugTest                             14343            20.8
DrugTests_THC_Positive                            15254            15.8
DrugTests_Cocaine_Positive                        15254            15.8
DrugTests_Meth_Positive                           15254            15.8
DrugTests_Other_Positive                          15254            15.8
Percent_Days_Employed                             17649             2.6
Jobs_Per_Year                                     17303             4.5
Employment_Exempt                                 18111             0.0
Recidivism_Within_3years                          18111             0.0
Recidivism_Arrest_Year1                           18111             0.0
Recidivism_Arrest_Year2                           18111             0.0
Recidivism_Arrest_Year3                           18111             0.0
Training_Sample                                   18111             0.0
```

```
ff_glimpse(recidivism)$Categorical[,c('n','missing_percent')]
```

```
                         n missing_percent
Gender                 18111             0.0
Race                   18111             0.0
Age_at_Release         18111             0.0
Supervision_Level_First 16962            6.3
Education_Level        18111             0.0
Prison_Offense         15820            12.6
Prison_Years           18111             0.0
```

We will focus a few ideas about how to handle missing data.

5.1. Creating missing data indicator variables

For every variable, we can create a binary indicator variable to indicate missingess (0: not missing, 1: missing). This doesn't solve the missing data problem because we may still have to impute the missing values for modeling; however, the indicator variables about whether or not variables are missing may sometimes provide some information in predicting the outcome. If the missingness is not random and there is a systematic relationship between outcome and whether or not values are missing for a variable, then this may be some important information to bring into the model. For variables that doesn't have any missing value, this indicator variable would be meaningless. Therefore, one can remove them from any further consideration.

```r
# The code below generates a separate corresponding matrix of the indicator
# variables for missingness in the original data matrix

missing_indicator <- function(x){
  ifelse(is.na(x),1, 0)
}

recidivism_missing   <- apply(X      = recidivism,
                              MARGIN = 2,
                              FUN    = missing_indicator)


# eliminate the variables with no missingness

  # Calculate the variance

  variance.missing <- apply(X      = recidivism_missing,
                            MARGIN = 2,
                            FUN    = var)

  # Find variable with non-zero variance

  var.no.missing <- which(variance.missing!=0)

  # Selec the variables with non-zero variance

  recidivism_missing <- recidivism_missing[,var.no.missing]

head(recidivism_missing)
```

```
     Gang_Affiliated Supervision_Risk_Score_First Supervision_Level_First
[1,]               0                           0                       0
[2,]               0                           0                       0
[3,]               0                           0                       0
[4,]               0                           0                       0
[5,]               0                           0                       0
[6,]               0                           0                       0
     Prison_Offense Avg_Days_per_DrugTest DrugTests_THC_Positive
[1,]              0                     0                      0
[2,]              0                     0                      0
[3,]              0                     0                      0
[4,]              0                     0                      0
[5,]              0                     0                      0
[6,]              1                     0                      0
     DrugTests_Cocaine_Positive DrugTests_Meth_Positive
[1,]                          0                       0
[2,]                          0                       0
[3,]                          0                       0
[4,]                          0                       0
[5,]                          0                       0
[6,]                          0                       0
     DrugTests_Other_Positive Percent_Days_Employed Jobs_Per_Year
[1,]                        0                     0            0
```

```
[2,]   0   0   0
[3,]   0   0   0
[4,]   0   0   0
[5,]   0   0   0
[6,]   0   0   0
```

5.2. Imputation

A common approach to missing data is to impute missing values. In imputation, each predictor becomes an outcome of interest and then the remaining predictors are used to build an imputation model to predict the missing values. Below is a very naive example of how it would work if we have an outcome variable (Y) and three predictors (X1, X2, X3). First, missing values are estimated and replaced for each predictor using an imputation model, and then the main outcome of interest is predicted using the imputed X1, X2, and X3.

Imputation Model	
Outcome	Predictors
X1	X2,X3
X2	X1,X3
X3	X1,X2

Prediction Model	
Outcome	Predictors
Y	X1, X2, X3

An imputation model can be something as simple as an intercept-only model (mean imputation). For numeric variables, missing values can be replaced with a simple mean, median, or mode of the observed data. For categorical variables, missing values can be replaced with a value randomly drawn from a binomial or multinomial distribution with the observed probabilities.

An imputation model can also be as complex as desired using a regularized regression model, a decision tree model, or K-nearest neighbors model. The main idea of a more complex prediction model is to find other observations as similar as to an observation with a missing value in terms of other predictors, and use data from these similar observations to predict the missing values. We will rely on some built-in functions in R to impute values using such complex models. For more information about its theoretical foundations, Applied Missing Data Analysis by Craig Enders provide a comprehensive coverage for this topic.

6. Wrapping-up using the `recipes` package

We can pre-process all the variables in the dataset manually using the approaches discussed earlier. However, this would be a tedious work. It may be overwhelming to apply all the procedures simulatenously to different versions of the datasets (e.g., training, test, future). Instead, we can use the `recipes` package to implement these approaches in a more organized an efficient way.

Before using the `recipes` package, there are a few things to do for this dataset.

1) Read the original data
2) Make a list of variables with different characteristics (categorical, continuous, proportions, etc.)
3) Make sure the type of all categorical variables are either `character` or `factor`.

4) For variables that represent proportions, add/substract a small number to 0s/1s for logit transformation

```
# 1) Read the original data

  recidivism <- read.csv('https://raw.githubusercontent.com/uo-datasci-specialization/c4-ml-fall-2021/ma
                         header=TRUE)

  str(recidivism)
```

```
'data.frame':   18111 obs. of  54 variables:
 $ ID                                          : int  1 2 3 4 6 7 8 11 13 15 ...
 $ Gender                                      : chr  "M" "M" "M" "M" ...
 $ Race                                        : chr  "BLACK" "BLACK" "BLACK" "WHITE" ...
 $ Age_at_Release                              : chr  "43-47" "33-37" "48 or older" "38-42" ...
 $ Residence_PUMA                              : int  16 16 24 16 17 18 16 5 18 5 ...
 $ Gang_Affiliated                             : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Supervision_Risk_Score_First                : int  3 6 7 7 5 2 5 3 3 7 ...
 $ Supervision_Level_First                     : chr  "Standard" "Specialized" "High" "High" ...
 $ Education_Level                             : chr  "At least some college" "Less than HS diploma
 $ Dependents                                  : int  3 1 3 1 0 2 3 1 1 1 ...
 $ Prison_Offense                              : chr  "Drug" "Violent/Non-Sex" "Drug" "Property" .
 $ Prison_Years                                : chr  "More than 3 years" "More than 3 years" "1-2
 $ Prior_Arrest_Episodes_Felony                : int  6 7 6 8 4 10 6 3 8 9 ...
 $ Prior_Arrest_Episodes_Misd                  : int  6 6 6 6 0 6 6 6 4 3 ...
 $ Prior_Arrest_Episodes_Violent               : int  1 3 3 0 1 1 3 2 0 2 ...
 $ Prior_Arrest_Episodes_Property              : int  3 0 2 3 3 5 1 1 5 2 ...
 $ Prior_Arrest_Episodes_Drug                  : int  3 3 2 3 0 1 2 1 2 4 ...
 $ Prior_Arrest_Episodes_PPViolationCharges    : int  4 5 5 3 0 5 5 3 1 4 ...
 $ Prior_Arrest_Episodes_DVCharges             : int  0 1 1 0 0 0 0 1 0 0 ...
 $ Prior_Arrest_Episodes_GunCharges            : int  0 0 0 0 0 1 0 0 0 1 ...
 $ Prior_Conviction_Episodes_Felony            : int  3 3 3 3 1 3 1 0 1 3 ...
 $ Prior_Conviction_Episodes_Misd              : int  3 4 2 4 0 1 4 3 0 2 ...
 $ Prior_Conviction_Episodes_Viol              : int  0 1 1 0 0 0 1 0 0 1 ...
 $ Prior_Conviction_Episodes_Prop              : int  2 0 1 3 2 3 0 0 2 1 ...
 $ Prior_Conviction_Episodes_Drug              : int  2 2 2 2 0 0 2 0 1 1 ...
 $ Prior_Conviction_Episodes_PPViolationCharges     : int  0 1 0 0 0 1 1 1 0 1 ...
 $ Prior_Conviction_Episodes_DomesticViolenceCharges: int  0 1 1 0 0 0 0 0 0 0 ...
 $ Prior_Conviction_Episodes_GunCharges        : int  0 1 0 0 0 1 0 0 0 0 ...
 $ Prior_Revocations_Parole                    : int  0 0 0 0 0 0 0 1 0 0 ...
 $ Prior_Revocations_Probation                 : int  0 0 0 1 0 0 0 0 0 0 ...
 $ Condition_MH_SA                             : int  1 0 1 1 0 0 0 1 0 1 ...
 $ Condition_Cog_Ed                            : int  1 0 1 1 0 0 1 1 0 1 ...
 $ Condition_Other                             : int  0 0 0 0 1 0 0 0 0 1 ...
 $ Violations_ElectronicMonitoring             : int  0 0 0 0 0 0 0 1 0 0 ...
 $ Violations_Instruction                      : int  0 1 1 0 0 0 0 1 0 0 ...
 $ Violations_FailToReport                     : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Violations_MoveWithoutPermission            : int  0 0 1 0 1 0 0 0 0 0 ...
 $ Delinquency_Reports                         : int  0 4 4 0 0 0 0 0 0 0 ...
 $ Program_Attendances                         : int  6 0 6 6 0 0 0 9 0 6 ...
 $ Program_UnexcusedAbsences                   : int  0 0 0 0 0 0 0 2 0 0 ...
 $ Residence_Changes                           : int  2 2 0 3 3 1 0 2 1 1 ...
 $ Avg_Days_per_DrugTest                       : num  612 35.7 93.7 25.4 474.6 ...
 $ DrugTests_THC_Positive                      : num  0 0 0.333 0 0 ...
 $ DrugTests_Cocaine_Positive                  : num  0 0 0 0 0 0 0 0 0 0 ...
```

```
$ DrugTests_Meth_Positive                          : num  0 0 0.167 0 0 ...
$ DrugTests_Other_Positive                         : num  0 0 0 0 0 ...
$ Percent_Days_Employed                            : num  0.489 0.425 0 1 0.674 ...
$ Jobs_Per_Year                                    : num  0.448 2 0 0.719 0.308 ...
$ Employment_Exempt                                : int  0 0 0 0 0 0 0 1 0 1 ...
$ Recidivism_Within_3years                         : int  0 1 1 0 0 1 0 1 0 0 ...
$ Recidivism_Arrest_Year1                          : int  0 0 0 0 0 0 0 0 0 0 ...
$ Recidivism_Arrest_Year2                          : int  0 0 1 0 0 0 0 1 0 0 ...
$ Recidivism_Arrest_Year3                          : int  0 1 0 0 0 1 0 0 0 0 ...
$ Training_Sample                                  : int  1 1 1 1 0 1 0 1 1 0 ...
```

```
# 2) List of variable types

  outcome <- c('Recidivism_Arrest_Year2')

  id      <- c('ID')

  categorical <- c('Residence_PUMA',
                   'Prison_Offense',
                   'Age_at_Release',
                   'Supervision_Level_First',
                   'Education_Level',
                   'Prison_Years',
                   'Gender',
                   'Race',
                   'Gang_Affiliated',
                   'Prior_Arrest_Episodes_DVCharges',
                   'Prior_Arrest_Episodes_GunCharges',
                   'Prior_Conviction_Episodes_Viol',
                   'Prior_Conviction_Episodes_PPViolationCharges',
                   'Prior_Conviction_Episodes_DomesticViolenceCharges',
                   'Prior_Conviction_Episodes_GunCharges',
                   'Prior_Revocations_Parole',
                   'Prior_Revocations_Probation',
                   'Condition_MH_SA',
                   'Condition_Cog_Ed',
                   'Condition_Other',
                   'Violations_ElectronicMonitoring',
                   'Violations_Instruction',
                   'Violations_FailToReport',
                   'Violations_MoveWithoutPermission',
                   'Employment_Exempt')

  numeric   <- c('Supervision_Risk_Score_First',
                 'Dependents',
                 'Prior_Arrest_Episodes_Felony',
                 'Prior_Arrest_Episodes_Misd',
                 'Prior_Arrest_Episodes_Violent',
                 'Prior_Arrest_Episodes_Property',
                 'Prior_Arrest_Episodes_Drug',
                 'Prior_Arrest_Episodes_PPViolationCharges',
                 'Prior_Conviction_Episodes_Felony',
                 'Prior_Conviction_Episodes_Misd',
                 'Prior_Conviction_Episodes_Prop',
```

```
                'Prior_Conviction_Episodes_Drug',
                'Delinquency_Reports',
                'Program_Attendances',
                'Program_UnexcusedAbsences',
                'Residence_Changes',
                'Avg_Days_per_DrugTest',
                'Jobs_Per_Year')

  props      <- c('DrugTests_THC_Positive',
                  'DrugTests_Cocaine_Positive',
                  'DrugTests_Meth_Positive',
                  'DrugTests_Other_Positive',
                  'Percent_Days_Employed')

# 3) Convert all nominal, ordinal, and binary variables to factors
  # Leave the rest as is

  for(i in categorical){

    recidivism[,i] <- as.factor(recidivism[,i])

  }

# 4) For variables that represent proportions, add/substract a small number
  # to 0s/1s for logit transformation

  for(i in props){
    recidivism[,i] <- ifelse(recidivism[,i]==0,.0001,recidivism[,i])
    recidivism[,i] <- ifelse(recidivism[,i]==1,.9999,recidivism[,i])
  }
```

Now, we can apply certain transformations to different types of variables. We will use the `step_***()` functions from the `recipes` package to implement different procedures. Below is a list of functions for most procedures discussed earlier in this lecture. A more detailed list of `step_***()` functions can be found the in the package manual.

- `step_dummy()`: creates dummy variables for one-hot encoding of categorical variables
- `step_indicate_na()` creates an indicator variable for missingness
- `step_impute_bag()`, `step_impute_knn()`, `step_impute_linear()`, and `step_impute_mean()`: imputes missing values using an imputation model
- `step_BoxCox()`: transforms non-negative data using Box-Cox method
- `step_poly`,`step_bs()`, and `step_ns()` : creates basis expansions
- `step_logit`: applies logit transformation
- `step_zv`: removes variables with zero variance
- `step_normalize`: standardize variables to have a mean of zero and standard deviation of one

Note that the order of procedures applied to variables is important. For instance, there would be no meaning of using `step_indicate_na()` after using `step_impute_bag()` (Why?). Or, there will be a problem when you first standardize variables using 'step_normalize()' and then apply a Box-Cox transformation (Why?).

For this dataset, we will implement the following steps:

1) Create an indicator variable of missingness for all predictors (`step_indicate_na`)

2) Remove the variables with zero variance (`step_zv`)
3) Impute the missing values for all predictor variables using a mean or mode (`step_impute_mean` and `step_impute_mode`)
4) Logit transform the variables that represent proportions(`step_logit`)
5) Create natural splines for all numeric variables (`step_ns`)
6) Standardize numeric features
7) One-hot encoding of all categorical variables (`step_dummy`)

```
require(recipes)

blueprint <- recipe(x  = recidivism,
                    vars  = c(categorical,numeric,props,outcome,id),
                    roles = c(rep('predictor',48),'outcome','ID')) %>%

  # for all 48 predictors, create an indicator variable for missingness

  step_indicate_na(all_of(categorical),all_of(numeric),all_of(props)) %>%

  # Remove the variable with zero variance, this will also remove the missingness
  # variables if there is no missingess

  step_zv(all_numeric()) %>%

  # Impute the missing values using mean and mode. You can instead use a
  # more advanced imputation model such as bagged trees. I haven't used it due
  # to time concerns

  step_impute_mean(all_of(numeric),all_of(props)) %>%
  step_impute_mode(all_of(categorical)) %>%

  #Logit transformation of proportions

  step_logit(all_of(props)) %>%

  # Natural splines for numeric variables and proportions

  step_ns(all_of(numeric),all_of(props),deg_free=3) %>%

  # Standardize the natural splines of numeric variables and proportions

  step_normalize(paste0(numeric,'_ns_1'),
                 paste0(numeric,'_ns_2'),
                 paste0(numeric,'_ns_3'),
                 paste0(props,'_ns_1'),
                 paste0(props,'_ns_2'),
                 paste0(props,'_ns_3')) %>%

  # One-hot encoding for all categorical variables

  step_dummy(all_of(categorical),one_hot=TRUE)



blueprint
```

```
Recipe

Inputs:

      role #variables
        ID          1
   outcome          1
 predictor         48

Operations:

Creating missing data variable indicators for all_of(categorical), all_of(numeric), all_of(pr...
Zero variance filter on all_numeric()
Mean Imputation for all_of(numeric), all_of(props)
Mode Imputation for all_of(categorical)
Logit transformation on all_of(props)
Natural Splines on all_of(numeric), all_of(props)
Centering and scaling for paste0(numeric, "_ns_1"), paste0(numeric, "_ns_...
Dummy variables from all_of(categorical)
```

Once the recipe is ready, we can train the blueprint on training data. When we say 'train' it means that weights or statistics for certain types of operations (e.g., standardization) are calculated based on training data and saved for later use. For instance, the mean and standard deviation of variable X from training data is calculated and later used to standardize the same variable X in testing data or future datasets. For now, we will train the blueprint using the whole dataset. In future lectures, we will split data into different subsets (training vs. test), and use the training dataset to apply the blueprint.

```
prepare <- prep(blueprint,
                training = recidivism)
prepare
```

```
Recipe

Inputs:

      role #variables
        ID          1
   outcome          1
 predictor         48

Training data contained 18111 data points and 7977 incomplete rows.

Operations:

Creating missing data variable indicators for Residence_PUMA, Prison_Offense, Age_at_Release,... [traine
Zero variance filter removed na_ind_Residence_PUMA, na_ind_Age_at_Re... [trained]
Mean Imputation for Supervision_Risk_Score_First, Dependents, Prior... [trained]
Mode Imputation for Residence_PUMA, Prison_Offense, Age_at_Release,... [trained]
Logit transformation on DrugTests_THC_Positive, DrugTests_Cocaine_Po... [trained]
Natural Splines on Supervision_Risk_Score_First, Dependents, Prior_A... [trained]
Centering and scaling for Supervision_Risk_Score_First_ns_1, Dependents_n... [trained]
Dummy variables from Residence_PUMA, Prison_Offense, Age_at_Release, Supervisi... [trained]
```

Finally, we can apply this recipe to our training and test datasets to obtain processed variables accordingly to use in modeling later.

```
baked_recidivism <- bake(prepare, new_data = recidivism)

baked_recidivism
```

```
# A tibble: 18,111 x 167
   Recidivism_Arrest_Year2    ID na_ind_Prison_O~ na_ind_Supervis~ na_ind_Gang_Aff~
                     <int> <int>            <int>            <int>            <int>
 1                       0     1                0                0                0
 2                       0     2                0                0                0
 3                       1     3                0                0                0
 4                       0     4                0                0                0
 5                       0     6                0                0                0
 6                       0     7                1                0                0
 7                       0     8                0                0                0
 8                       1    11                0                0                0
 9                       0    13                0                0                0
10                       0    15                0                0                0
# ... with 18,101 more rows, and 162 more variables:
#   na_ind_Supervision_Risk_Score_First <int>,
#   na_ind_Avg_Days_per_DrugTest <int>, na_ind_Jobs_Per_Year <int>,
#   na_ind_DrugTests_THC_Positive <int>,
#   na_ind_DrugTests_Cocaine_Positive <int>,
#   na_ind_DrugTests_Meth_Positive <int>,
#   na_ind_DrugTests_Other_Positive <int>, ...
```

Notice that there are 167 variables (one outcome, one ID, and 165 predictors) in the new processed datasets. In the original dataset, there were 48 predictors. Below is a breakdown of where these 165 variables come from.

Variable Name	Encoding	Number of Categories	Process	Number of Constructed Variables
Gender	Binary	2	One-hot encoding	2
Race	Binary	2	One-hot encoding	2
Gang affiliation	Binary	2	One-hot encoding	2
Prior_Arrest_Episodes_DVCharges	Binary	2	One-hot encoding	2
Prior_Arrest_Episodes_GunCharges	Binary	2	One-hot encoding	2
Prior_Conviction_Episodes_Viol	Binary	2	One-hot encoding	2
Prior_Conviction_Episodes_PPViolationCharges	Binary	2	One-hot encoding	2
Prior_Conviction_Episodes_DomesticViolenceCharges	Binary	2	One-hot encoding	2
Prior_Conviction_Episodes_GunCharges	Binary	2	One-hot encoding	2

Variable Name	Encoding	Number of Categories	Process	Number of Constructed Variables
Prior_Revocations_Parole	Binary	2	One-hot encoding	2
Prior_Revocations_Probation	Binary	2	One-hot encoding	2
Condition_MH_SA	Binary	2	One-hot encoding	2
Condition_Cog_Ed	Binary	2	One-hot encoding	2
Condition_Other	Binary	2	One-hot encoding	2
Violations_ElectronicMonitoring	Binary	2	One-hot encoding	2
Violations_Instruction	Binary	2	One-hot encoding	2
Violations_FailToReport	Binary	2	One-hot encoding	2
Violations_MoveWithoutPermission	Binary	2	One-hot encoding	2
Employment_Exempt	Binary	2	One-hot encoding	2
Age_at_Release	Ordinal	7	One-hot encoding	7
Supervision_Level_First	Ordinal	3	One-hot encoding	3
Education_Level	Ordinal	3	One-hot encoding	3
Prison_Years	Ordinal	4	One-hot encoding	4
Residence_PUMA	Nominal	25	One-hot encoding	25
Prison_Offense	Nominal	5	One-hot encoding	5
Supervision_Risk_Score_First	Numeric		Natural Splines, Standard-ization	3
Dependents	Numeric		Natural Splines, Standard-ization	3
Prior_Arrest_Episodes_Felony	Numeric		Natural Splines, Standard-ization	3

Variable Name	Encoding	Number of Categories	Process	Number of Constructed Variables
Prior_Arrest_Episodes_Misd	Numeric		Natural Splines, Standard-ization	3
Prior_Arrest_Episodes_Violent	Numeric		Natural Splines, Standard-ization	3
Prior_Arrest_Episodes_Property	Numeric		Natural Splines, Standard-ization	3
Prior_Arrest_Episodes_Drug	Numeric		Natural Splines, Standard-ization	3
Prior_Arrest_Episodes_PPViolationCharges	Numeric		Natural Splines, Standard-ization	3
Prior_Conviction_Episodes_Felony	Numeric		Natural Splines, Standard-ization	3
Prior_Conviction_Episodes_Misd	Numeric		Natural Splines, Standard-ization	3
Prior_Conviction_Episodes_Prop	Numeric		Natural Splines, Standard-ization	3
Prior_Conviction_Episodes_Drug	Numeric		Natural Splines, Standard-ization	3
Delinquency_Reports	Numeric		Natural Splines, Standard-ization	3
Program_Attendances	Numeric		Natural Splines, Standard-ization	3
Program_UnexcusedAbsences	Numeric		Natural Splines, Standard-ization	3
Residence_Changes	Numeric		Natural Splines, Standard-ization	3

Variable Name	Encoding	Number of Categories	Process	Number of Constructed Variables
Avg_Days_per_DrugTest	Numeric		Natural Splines, Standard-ization	3
Jobs_Per_Year	Numeric		Natural Splines, Standard-ization	3
DrugTests_THC_Positive	Numeric		Natural Splines, Standard-ization	3
DrugTests_Cocaine_Positive	Numeric		Natural Splines, Standard-ization	3
DrugTests_Meth_Positive	Numeric		Natural Splines, Standard-ization	3
DrugTests_Other_Positive	Numeric		Natural Splines, Standard-ization	3
Percent_Days_Employed	Numeric		Natural Splines, Standard-ization	3
Missing value indicator variables	Binary			11
			Total	165