8 - Handling Missing Data

Feature Engineering and Selection: A Practical Approach for Predictive Models

Emanuela Elli (892901), Alessandro Fasani (837301), Federica Madon (825628)

Librerie utilizzate

```
# Caricamento delle librerie e delle funzioni utili
library(RCurl)
library(caret)
library(tidyverse)
library(naniar)
library(visdat)
library(knitr)
require(skimr)
library(ComplexHeatmap)
library(rpart)
library(partykit)
library(tidymodels)
library(gridExtra)
library(lubridate)
library(ggiraph)
library(heatmaply)
library(RColorBrewer)
library(scales)
library(ipred)
library(recipes)

l10_breaks <- scales::trans_breaks("log10", function(x) 10^x)
l10_labels <- scales::trans_format("log10", scales::math_format(10^.x))
```

Tra tutte le librerie utilizzate quelle più importanti sono:

- `nanir` è un pacchetto molto utile per visualizzare e gestire i dati mancanti in R;
- `caret` (dal libro Kuhn, Johnson, 2013, Applied Predictive Modelling Springer), trattata anche durante il corso la lezione inerente alla regressione non parametrica e che contiene funzioni utili nel processo di modellazione predittiva;
- `rpart`, libreria di machine learning utilizzata per creare alberi di classificazione e regressione;
- `partykit`, toolkit flessibile per l'apprendimento, la rappresentazione, il riepilogo e la visualizzazione di un'ampia gamma di modelli di classificazione e regressione strutturati ad albero;
- Il pacchetto `recipes` prepara i dati per la modellazione;
- `ipred` è una libreria che contiene funzioni per migliorare i modelli predittivi mediante la classificazione indiretta e il bagging per problemi di classificazione, regressione e sopravvivenza nonché stimatori basati sul ricampionamento dell'errore di previsione.

Datasets utilizzati

I datasets utilizzati si possono trovare al seguente link: https://github.com/topepo/FES/tree/master/Data_Sets/Chicago_trains.

```
#Load dei dataset necessari
data(scat)
load("C:/Users/madon/OneDrive/Desktop/Hdda/chicago.RData")
load("C:/Users/madon/OneDrive/Desktop/Hdda/chicago_raw_entries.RData")
load("C:/Users/madon/OneDrive/Desktop/Hdda/stations.RData")
```

Per le visualizzazioni verranno utilizzati due fonti di dati: *Scat data* e *Chicago Train Ridership data*. *Scat data* è un dataset contenuto nel pacchetto `caret` e conta 110 osservazioni circa test di laboratorio e morfologici su campioni di escrementi di animali trovati in natura. L'obiettivo alla base della raccolta era focalizzato sul trovare una corrispondenza tra le misure relative ai predittori e la specie che ha prodotto l'escremento (preventivamente classificato tramite il genotipo nel DNA). Di quelle 110 osservazioni, 19 presentano uno o più valori mancanti. La raccolta di datasets di *Chicago Train Ridership* contiene 3 dataset di nostro interesse. Uno di questi, `raw_entries`, riporta il numero (indicato in migliaia) di passeggeri presenti in 5.733 giorni (osservazioni) e lungo 137 stazioni (predittori). Queste sono state ottenute misurando il numero di ingressi in una stazione in tutti i tornelli. I valori variano tra 0 e 36.323 al giorno. Le stazioni, nonché predittori di interesse, sono state tutte codificate tramite un id univoco. Per questa ragione è stato utilizzato anche il dataset `stations` al fine di assegnare a ciascun id il nome proprio della stazione che rappresenta.

```
head(scat[, c("Species", "Length", "Diameter", "Taper", "TI","Mass","d13C",
              "d15N", "CN", "flat")])
```

```
## # A tibble: 6 x 10
##   Species Length Diameter Taper    TI  Mass  d13C  d15N    CN  flat
##   <fct>    <dbl>    <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <int>
## 1 coyote     9.5     25.7  41.9  1.63  15.9 -26.8  6.94   8.5     0
## 2 coyote    14       25.4  37.1  1.46  17.6 -29.6  9.87  11.3     0
## 3 bobcat     9       18.8  16.5  0.88   8.4 -28.7  8.52   8.1     0
## 4 coyote     8.5     18.1  24.7  1.36   7.4 -20.1  5.79  11.5     0
## 5 coyote     8       20.7  20.1  0.97  25.4 -23.2  7.01  10.6     0
## 6 coyote     9       21.2  28.5  1.34  14.1 -29    8.28   9       0
```

```
Range = function(x) {
  a = range(x, na.rm = TRUE)
  a[2]-a[1]
}


PerSkim = skim_with(numeric = sfl(hist = NULL, sd = NULL, p25 = NULL,
                                  p50 = NULL, p75 = NULL, range = Range))
PerSkim(scat[, c("Species", "Length", "Diameter", "Taper", "TI","Mass","d13C",
                 "d15N", "CN", "flat")])
```

Table 1: Data summary

Name	. . . []
Number of rows	110

Table 1: Data summary

Number of columns	10
Column type frequency:	
factor	1
numeric	9
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Species	0	1	FALSE	3	bob: 57, coy: 28, gra: 25

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	p0	p100	range
Length	0	1.00	9.30	2.50	20.50	18.00
Diameter	6	0.95	18.56	7.80	30.00	22.20
Taper	17	0.85	27.43	2.30	91.50	89.20
TI	17	0.85	1.60	0.23	8.68	8.45
Mass	1	0.99	12.46	0.94	53.70	52.76
d13C	2	0.98	-26.86	-29.85	-19.67	10.18
d15N	2	0.98	7.44	1.84	18.00	16.16
CN	2	0.98	8.40	4.50	23.60	19.10
flat	0	1.00	0.05	0.00	1.00	1.00

Di seguito il dataset `raw_missing`:

```
head(raw_entries[, c(1:10)])
```

```
## # A tibble: 6 x 10
##   date       s_40010 s_40020 s_40030 s_40040 s_40050 s_40060 s_40070 s_40080
##   <date>       <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 2001-01-01   0.29    0.633   0.483   0.374   0.804    1.16   0.649    1.12
## 2 2001-01-02   1.24    2.95    1.23    7.74    3.20     4.05   5.78     3.85
## 3 2001-01-03   1.41    3.11    1.39    8.05    3.48     4.15   6.48     4.15
## 4 2001-01-04   1.39    3.26    1.37    8.03    3.54     4.36   6.77     4.20
## 5 2001-01-05   1.46    3.36    1.45    7.65    3.68     4.4    6.31     4.40
## 6 2001-01-06   0.613   1.57    0.839   0.844   2.47     2.23   1.80     2.54
## # i 1 more variable: s_40090 <dbl>
```

```
with.NaN = vapply(raw_entries[, -1], function(x) sum(is.na(x)) > 0, logical(1))
with.NaN.names = names(which(with.NaN==TRUE))

PerSkim(raw_entries[, c(with.NaN.names)])
```

Table 4: Data summary

Name	raw_entries[, c(with.NaN....
Number of rows	5733
Number of columns	15
Column type frequency:	
numeric	15
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	p0	p100	range
s_40190	30	0.99	3.93	0	13.83	13.83
s_40240	30	0.99	6.55	0	11.82	11.82
s_40450	30	0.99	10.87	0	18.90	18.90
s_40500	2780	0.52	4.85	0	12.44	12.44
s_40910	30	0.99	2.93	0	9.55	9.55
s_40990	30	0.99	5.12	0	8.74	8.74
s_41000	30	0.99	3.48	0	11.70	11.70
s_41170	30	0.99	3.45	0	5.99	5.99
s_41230	30	0.99	2.63	0	7.39	7.39
s_41430	30	0.99	4.21	0	9.67	9.67
s_41510	4138	0.28	1.89	0	4.28	4.28
s_41580	5702	0.01	0.00	0	0.03	0.03
s_41670	151	0.97	0.70	0	1.84	1.84
s_41680	4108	0.28	0.67	0	1.26	1.26
s_41690	5113	0.11	1.12	0	3.55	3.55

Comprendere la natura e la gravità delle informazioni mancanti

Per i successivi 3 esempi di visualizzazioni verrà utilizzato il dataset *Scat Data*.

Creiamo un dataset binario codificando con 0 i valori mancanti e con 1 tutti gli altri valori possibili:

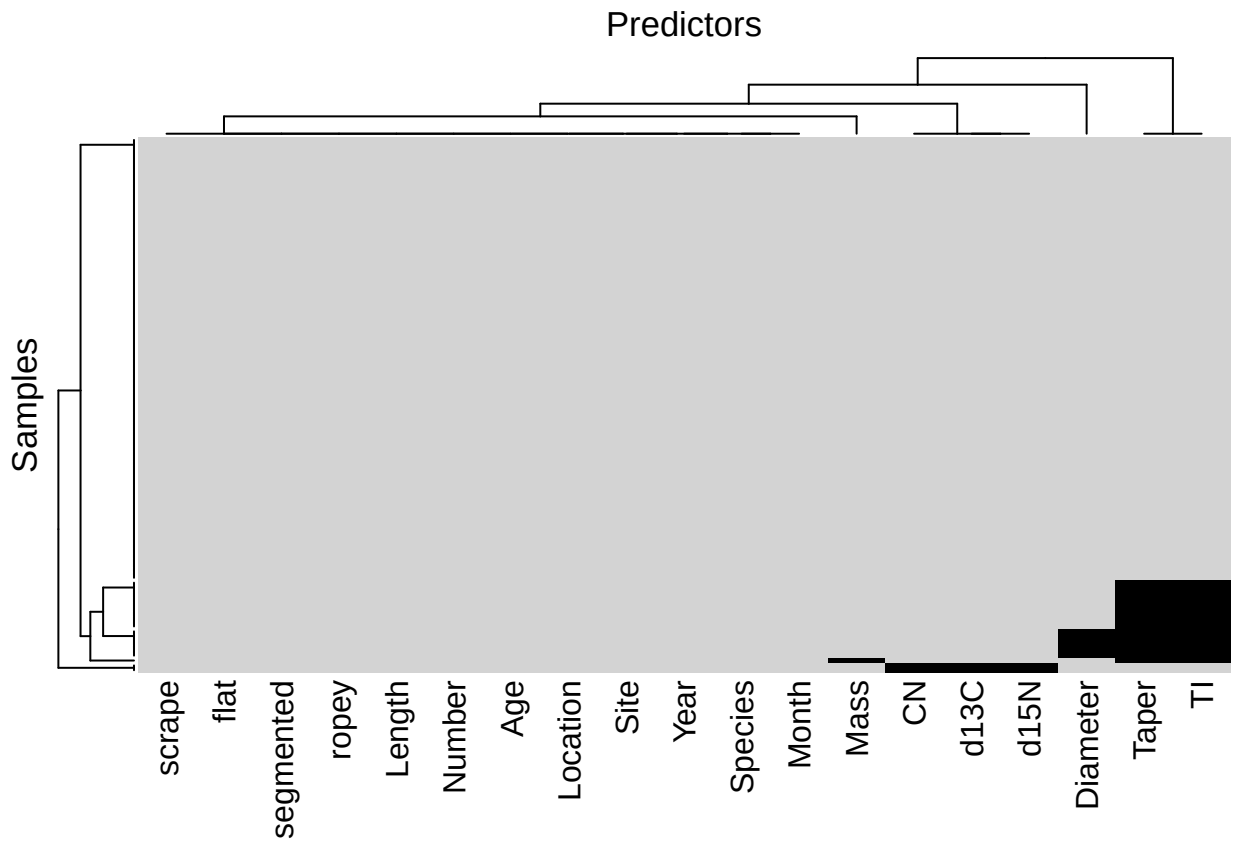
```
convert_missing <- function(x) ifelse(is.na(x), 0, 1)
scat_missing <- apply(scat, 2, convert_missing)
```

Heatmaps

Le *Heatmaps* sono un ottimo modo per visualizzare la natura delle informazioni mancanti quando i datasets sono di dimensioni ridotte. Si procede distinguendo binariamente tutti i valori del dataset a seconda che siano presenti valori NA o meno.

```
Heatmap(
  scat_missing,
  name = "Missing", #title of legend
  column_title = "Predictors", row_title = "Samples",
```

```
  col = c("black","lightgrey"),
  show_heatmap_legend = FALSE,
  row_names_gp = gpar(fontsize = 0) # Text size for row names
)
```



I dendrogrammi alle estremità dell'heatmap visualizzano in ordine decrescente di intensità, la presenza di missing values tra le covariate (asse X) e tra le singole osservazioni (asse Y), posizionandole vicine tra loro ed in ordine:

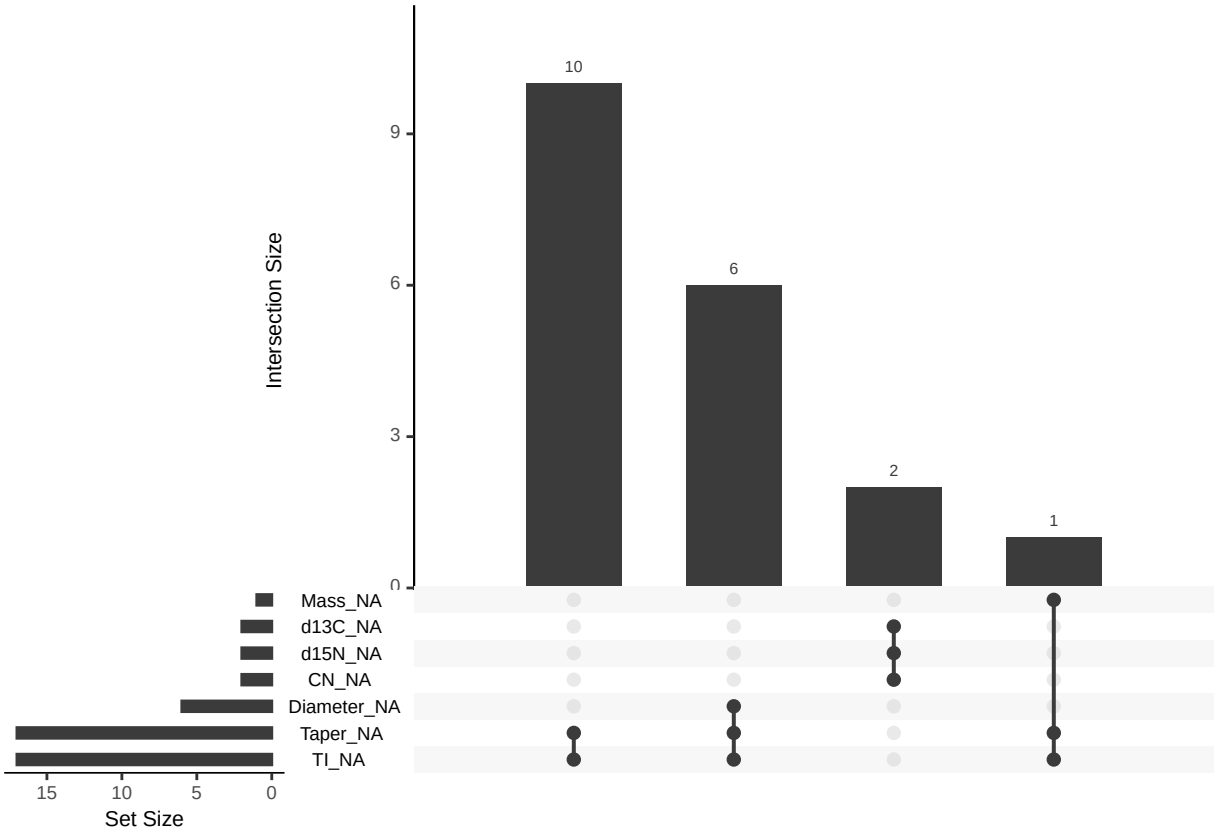
- Decrescente da destra a sinistra le covariate con più osservazioni mancanti;
- Decrescente dal basso verso l'alto le osservazioni con più variabili mancanti.

Nativamente l'heatmap riordina secondo un algoritmo di clustering per similarità.

Co-occurence plot

Il *Co-occurence plot* si focalizza sui predittori mancanti, evidenziando quali di questi e quante volte mancano congiuntamente tra le osservazioni del campione.

```
gg_miss_upset(scat, nsets = 7)
```

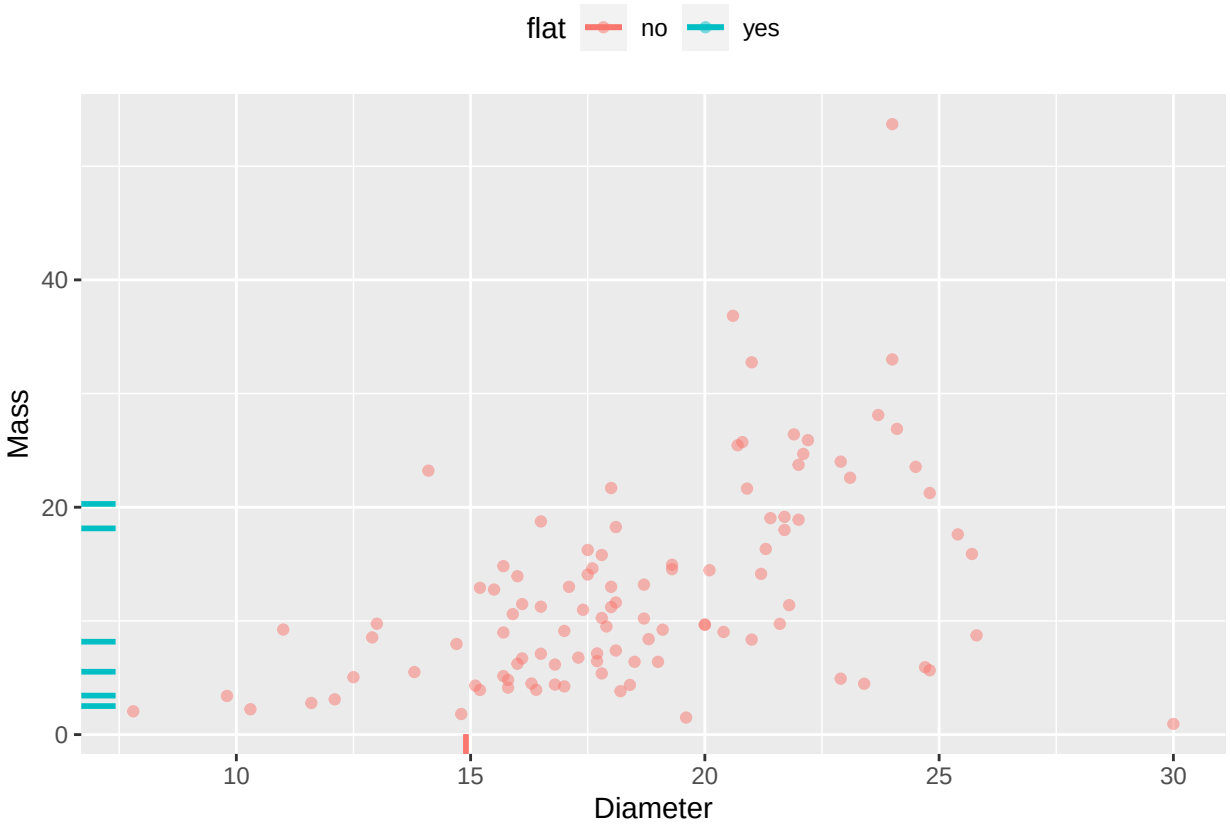


La funzione `gg_miss_upset()` del pacchetto `nanir` mostra un co-occurence plot dei predittori e il numero di osservazioni che presentano queste combinazioni di predittori mancanti, utilizzando rispettivamente una sintesi grafica di queste variabili e un grafico a barre.

Scatterplot

Gli *scatterplot* possono essere un ottimo modo per visualizzare le relazioni tra i dati e i valori mancanti. In particolare, può essere analizzata l'interazione dei valori mancanti tra due variabili numeriche, alla luce di una terza variabile categorica (binaria in questo caso).

```
scat_flat = mutate(scat, flat = ifelse(flat == 1, "yes", "no"))

ggplot(scat_flat, aes(col = flat)) +
  geom_point(aes(x = Diameter, y = Mass), alpha = .5) +
  geom_rug(data = scat_flat[is.na(scat_flat$Mass),],
           aes(x = Diameter),
           sides = "b",
           lwd = 1) +
  geom_rug(data = scat_flat[is.na(scat_flat$Diameter),],
           aes(y = Mass),
           sides = "l",
           lwd = 1) +
  theme(legend.position = "top")
```



Dopo aver mutato la variabile `flat` in binaria `yes-->1`, `no-->0` vengono utilizzate diverse funzioni della libreria `ggplot2`. La funzione `ggplot()` per inizializzare lo scatterplot di base di `ggplot2` e differenziare per colore la variabile binaria `flat`, `geom_point()` per i singoli punti/osservazioni, `geom_rug` per evidenziare con delle linee più marcate la presenza di missing values sui due assi cartesiani.

Quando il numero di osservazioni e/o predittori non è più contenuto in poche centinaia, le precedenti visualizzazioni risultano di difficile comprensione e non permettono di individuare eventuali pattern presenti nei dati.

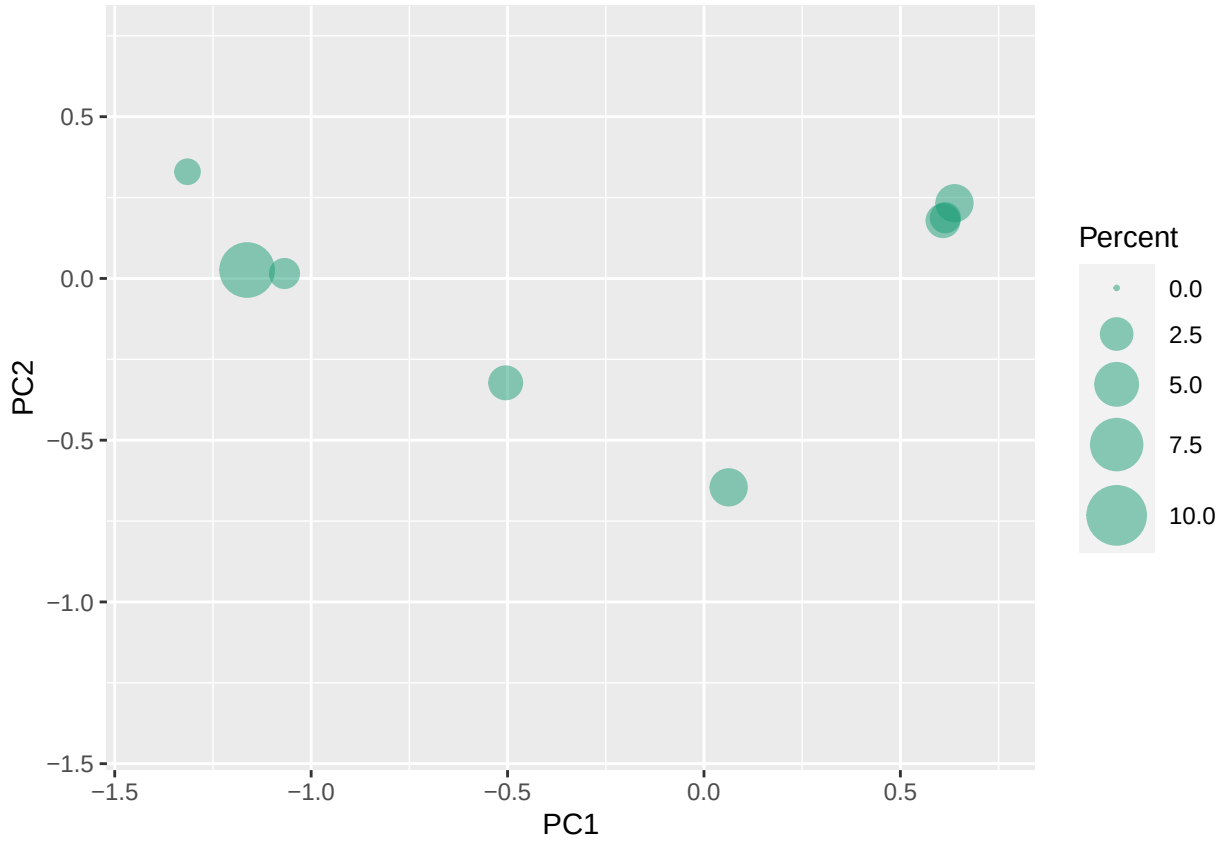
PCA

Per i successivi 3 esempi di visualizzazioni e per le tabelle di sintesi verrà utilizzato il dataset *Chicago train ridership*.

La *Principal Component Analysis* è una tecnica di riduzione della dimensionalità che può risultare molto utile per visualizzare nel complesso il dataset e i predittori con valori mancanti. Lo scopo della PCA è individuare le direzioni con i massimi valori di variabilità e individuare il numero di pattern presenti tra i dati mancanti.
Nel successivo plot sarà focalizzata l'attenzione sulle osservazioni.

```
only_rides = dplyr::select(raw_entries, -date)

date_missing = apply(only_rides, 2, convert_missing)

#Vector containing the % of missing values for each observation (5733)
```

```r
date_missing_ppn = apply(only_rides, MARGIN = 1, function(x) sum(is.na(x)))/ncol(only_rides)

#PCA
pca_dates <- prcomp(date_missing)

pca_d =
  data.frame(pca_dates$x) %>%
  dplyr::select(PC1, PC2) %>%
  mutate(Percent = date_missing_ppn * 100) %>%
  dplyr::distinct(PC1, PC2, Percent)

pca_d_rng <- extendrange(c(pca_d$PC1, pca_d$PC2))

ggplot(pca_d, aes(x = PC1, y = PC2, size = Percent)) +
  geom_point(alpha = .5, col = "#1B9E77") +
  xlim(pca_d_rng) +
  ylim(pca_d_rng) +
  scale_size(limits = c(0, 10), range = c(.5, 10))
```

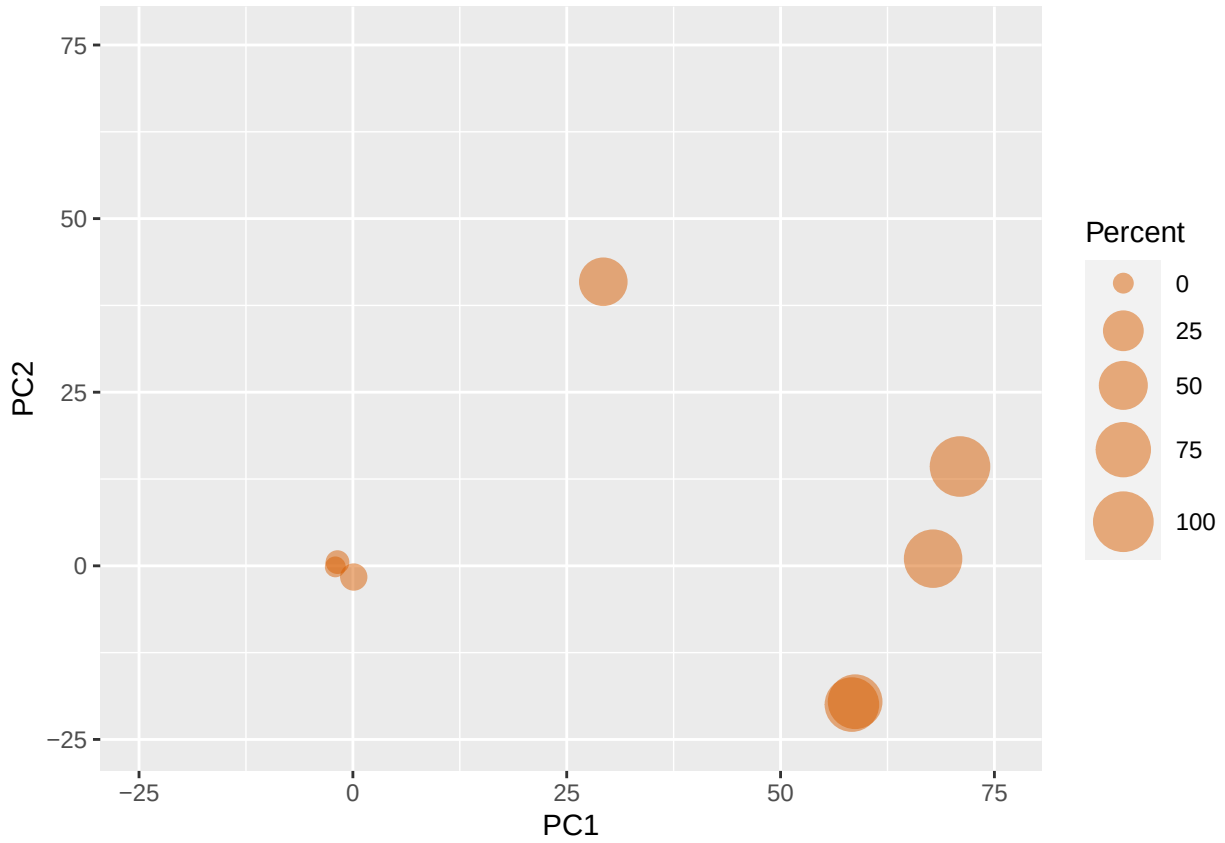


Dalla tabella delle `raw_entities` selezioniamo solo le colonne relative alle stazioni utilizzando la funzione `select()` e da questa creiamo una nuova tabella `date_missing` codificando binariamente tra valore presente e valore assente, usando la funzione precedentemente creata `convert_missing()`. Creiamo un vettore `date_missing_ppn` contenente la proporzione di missing values per ciascuna delle 5733 osservazioni, rispetto al totale dei predittori. Applichiamo la PCA alla tabella con valori binari, tramite la funzione `prcomp()`, per ottenere un oggetto contenente *Eigenvalues* e *Eigenvectors*. Da questo oggetto selezioniamo le prime due componenti, PC1 e PC2 e aggiungiamo alla tabella creata una nuova colonna `Percent` contenente le % di

missing values per ogni osservazione, utilizzando la funzione `mutate()`. Salviamo in una variabile `pca_d_rng` i valori dei range di dati delle due componenti PC1 e PC2, estesi di una piccola percentuale, per ottenere un plot migliore visualizzando tutti i punti. Come ultima fase viene stampato tutto con la funzione `ggplot()` evidenziando le due componenti delle PCA sui due assi cartesiani e le dimensioni dei pattern (sfruttando la colonna `Percent` appena creata) riscontrati tra le osservazioni.

Nel successivo plot, sempre relativo alla PCA, sarà focalizzata l'attenzione sui predittori.

```
cols_missing <-
  only_rides %>%
  summarise_all(list(~ sum(is.na(.)))) %>%
  mutate_all(funs(ppn = ./nrow(only_rides))) %>%
  dplyr::select(ends_with("ppn")) %>%
  gather(key = "station_label", value = ppn) %>%
  mutate(station_label = gsub("_ppn", "", station_label))

#PCA
pca_stations <- prcomp(t(date_missing))

pca_s <-
  data.frame(pca_stations$x) %>%
  dplyr::select(PC1, PC2) %>%
  bind_cols(cols_missing) %>%
  mutate(Percent = ppn * 100) %>%
  dplyr::distinct(PC1, PC2, Percent, ppn)

pca_s_rng <- extendrange(c(pca_s$PC1, pca_s$PC2))


ggplot(pca_s, aes(x = PC1, y = PC2, size = Percent)) +
  geom_point(alpha = .5, col = "#D95F02") +
  xlim(pca_s_rng) +
  ylim(pca_s_rng) +
  scale_size_continuous(limits = c(0, 100), range = c(3, 10))
```



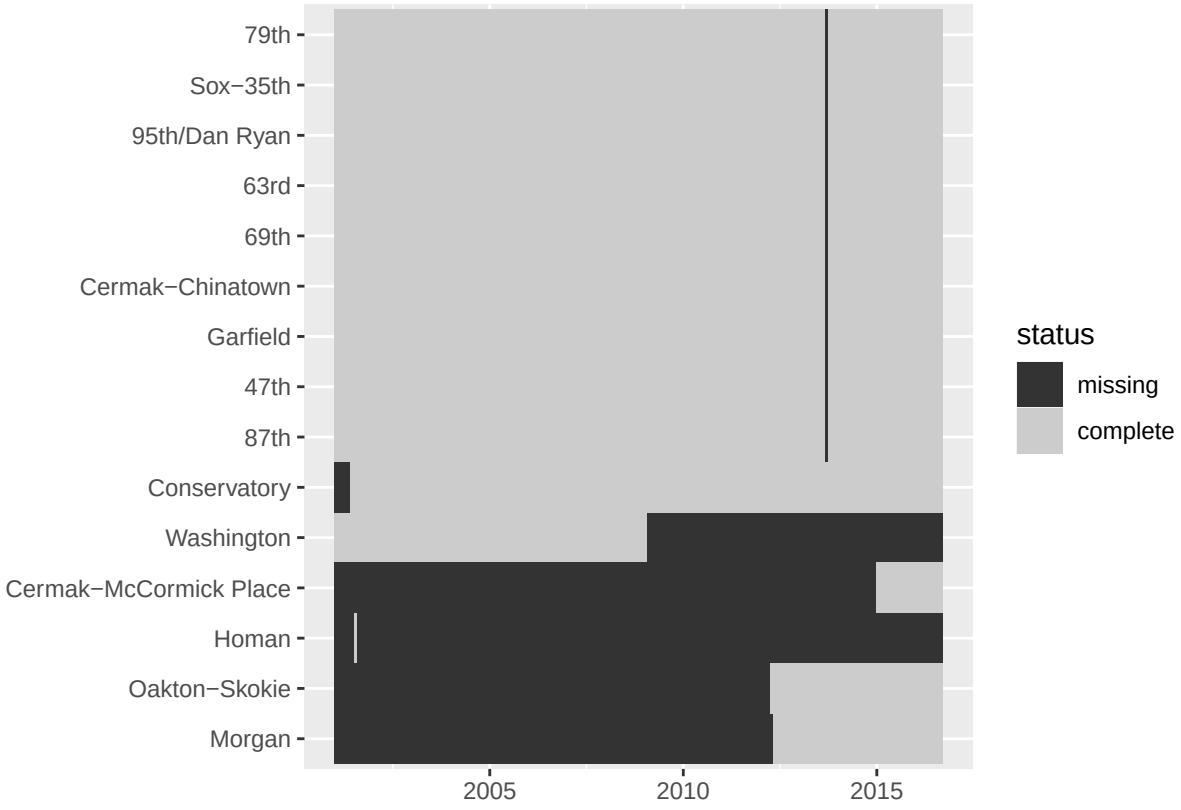
Dalla tabella `only_rides` creiamo una nuova tabella `cols_missing` contenente le proporzioni di valori mancanti per ogni colonna/stazione ferroviaria. Per la sua costruzione sono stati contati tutti i valori NaN perciascuna colonna, con la funzione `summarise_all()`, per poi essere mutati in proporzione rispetto al numero di righe/osservazioni tramite la funzione `mutate_all()`. Successivamente applichiamo la PCA alla trasposta della tabella `date_missing` con valori binari, tramite la funzione `prcomp()`, per ottenere un oggetto contenente *Eigenvalues* e *Eigenvectors*. Da questo oggetto selezioniamo le prime due componenti, PC1 e PC2 e aggiungiamo alla tabella creata una nuova colonna `Percent` contenente le % di missing values per ogni covariata/stazione, utilizzando la funzione `mutate()`. Salviamo in una variabile `pca_s_rng` i valori dei range di dati delle due componenti PC1 e PC2, estesi di una piccola percentuale, per ottenere un plot migliore visualizzando tutti i punti. Come ultima fase viene stampato tutto con la funzione `ggplot()` evidenziando le due componenti delle PCA sui due assi cartesiani e le dimensioni dei pattern (sfruttando la colonna `Percent` appena creata) riscontrati tra le covariate/stazioni.

Missing data patterns

Per avere un ulteriore comprensione di come la PCA individua dei pattern, può essere utile visualizzare una rappresentazione dei missing values delle stazioni (ordinate con algoritmi di clustering per distanza) e dei giorni (ordinati cronologicamente).

```
miss_entries <-
  raw_entries %>%
  dplyr::select(-date) %>%
  is.na()

miss_num <- apply(miss_entries, 2, sum)
```

```r
has_missing <- vapply(raw_entries[, -1], function(x) sum(is.na(x)) > 1, logical(1))
miss_station <- names(has_missing)[has_missing]


#Clustering on just the station data (not time) and get a reordering of the stations for plotting
miss_data <-
  raw_entries[, miss_station] %>%
  is.na()

clst <- hclust(dist(t(miss_data)))
clst_stations <- tibble(
    station_id = colnames(miss_data),
    order = clst$order)


station_names <-
  stations %>%
  dplyr::select(name, station_id) %>%
  right_join(clst_stations, by = "station_id")

station_lvl <- station_names[["name"]][station_names$order]

miss_vert <-
  raw_entries %>%
  gather(station_id, raw_entries, -date) %>%
  filter(station_id %in% miss_station) %>%
  mutate(status = ifelse(is.na(raw_entries), "missing", "complete")) %>%
  full_join(station_names, by = "station_id") %>%
  mutate(
    name = factor(name, levels = station_lvl),
    status = factor(status, levels = c("missing", "complete"))
  )


ggplot(miss_vert, aes(x = date, y = name, fill = status)) +
  geom_tile() +
  ylab("") + xlab("") +
  scale_fill_grey()
```



Tramite la funzione `vapply()` creiamo un vettore `has_missing` contenente `TRUE` o `FALSE` a seconda che ci siano o meno 2 o più valori mancanti, poi associamo ad una variabile `miss_station` solo quelle `TRUE` (15 in totale) tenendo solo i rispettivi `id_stazione`. Dopo aver creato la tabella `miss_data` con le sole colonne che presentavano due o più valori missing, si procede creando una tabella contenente gli id delle 15 stazioni ed un numero di ordine assegnato da un algoritmo di clustering gerarchico per distanza, fornito dalla funzione `hclust()` del pacchetto `stats`. Dopo aver creato una tabella `station_names`, contenente i nomi propri delle stazioni (right join sul dataset stations), si procede creando un ulteriore tabella `miss_vert` ordinando tutte le osservazioni raccolte, e relative alle sole stazioni del dataset `miss_station`, secondo l'ordine fornitoci dall'algoritmo di clustering. Il tutto viene poi plottato con la funzione `ggplot()`.

Misure di sintesi

Le misure di sintesi sono un altro valido strumento per comprendere la gravità dei missing data tra osservazioni e predittori. Tra le più semplici si annoverano le frequenze relative % di missing values nei predittori e le frequenze relative % di missing values nelle singole osservazioni (raggruppate per range o per medesima percentuale di missing values).

Percentuale di dati mancanti per i predittori:

```
summary_variabili = miss_var_summary(as.data.frame(scat))
summary_variabili$pct_miss = round(summary_variabili$pct_miss, digits = 1)
summary_variabili_grouped = group_by(summary_variabili, pct_miss)
svg_final = as.data.frame(attr(summary_variabili_grouped, "groups"))
colnames(svg_final) = c("% Dati mancanti", "id Feature")
svg_final=  kable(svg_final)
svg_final
```

% Dati mancanti	id Feature
0.0	8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19
0.9	7
1.8	4, 5, 6
5.5	3
15.5	1, 2

Utilizzando la funzione `miss_var_summary()` creiamo una tabella di sintesi per ogni predittore circa la percentuale di osservazioni mancanti che presenta. Con la funzione `group_by()` raggruppiamo insieme i predittori che presentano la stessa percentuale di osservazioni mancanti ed infine estraiamo dagli attributi della variabile `summary_variabili_grouped` appena creata, la tabella `grouped`, utilizzando la funzione `attr()`.

Percentuale di dati mancanti per le osservazioni:

```
summary_oss = miss_case_summary(as.data.frame(scat))
summary_oss$pct_miss = round(summary_oss$pct_miss, digits = 1)
summary_oss_grouped = group_by(summary_oss, pct_miss)
svo_final= as.data.frame(attr(summary_oss_grouped, "groups"))
colnames(svo_final) = c("% Dati mancanti", "id Osservazione")
svo_final = kable(svo_final)
svo_final
```

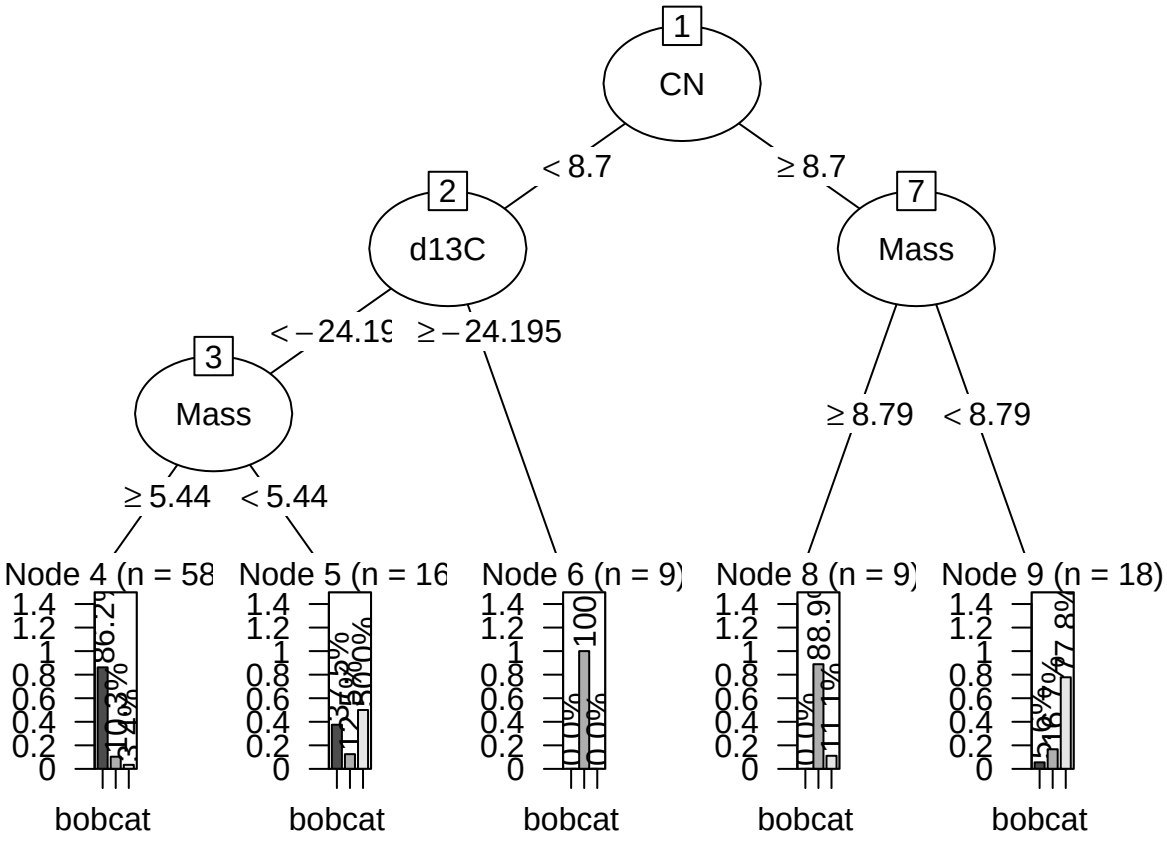
% Dati mancanti	id Osservazione
0.0	20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110
10.5	10, 11, 12, 13, 14, 15, 16, 17, 18, 19
15.8	1, 2, 3, 4, 5, 6, 7, 8, 9

Utilizzando la funzione `miss_case_summary()` creiamo una tabella di sintesi per ogni osservazione circa la % di predittori mancanti che presenta. Con la funzione `group_by()` raggruppiamo insieme le osservazioni che presentano la stessa % di predittori mancanti ed infine estraiamo dagli attributi della variabile `summary_oss_grouped` appena creata, la tabella `grouped`, utilizzando la funzione `attr()`.

Modelli "resistenti" ai valori mancanti

Nella seguente visualizzazione viene mostrato un albero di partizionamento ricorsivo per i dati relativi agli escrementi di animali (precedentemente già introdotti).
Per costruire l'albero di decisione secondo il modello CART viene utilizzata la funzione `rpart()` in cui si specifica che la variabile target da prevedere è `Species` e le variabili esplicative con cui si vuole effettuare tale previsione sono tutte le variabili presenti all'interno del dataset (indicato col simbolo `.`).
Dopodiché si procede alla conversione in un oggetto `party` con la funzione `as.party()` dal pacchetto `partykit`.

```
rpart_mod <- rpart(Species ~ ., data=scat)
rpart_party <- as.party(rpart_mod)
plot(rpart_party, tp_args = list(text = "vertical", ymax = 1.5))
```



Come viene mostrato di seguito, è possibile notare come i tre predittori utilizzati all'interno dell'albero di decisione (`CN`, `d13C`, `Mass`) contengono valori mancanti all'interno del dataset.

```
colSums(is.na(scat))
```

```
##   Species     Month      Year      Site  Location       Age    Number    Length
##         0         0         0         0         0         0         0         0
##  Diameter     Taper        TI      Mass      d13C      d15N        CN     ropey
##         6        17        17         1         2         2         2         0
## segmented      flat    scrape
##         0         0         0
```

Nel modello CART questo non risulta essere una problematica poichè quando si verifica la presenza di valori mancanti nei predittori vengono utilizzati in egual modo dei predittori surrogati (*surrogate splits*).
Pertanto per ogni divisione, vengono valutate una varietà di alternative e vengono considerate tali le suddivisioni i cui risultati sono simili alla suddivisione originale effettivamente utilizzata nell'albero. Se una suddivisione surrogata si avvicina bene alla suddivisione originale, può essere utilizzata quando i dati del predittore associati alla suddivisione originale non sono disponibili. Per cui fondamentalmente non solo viene memorizzato lo split migliore (chiamato split primario) ma anche diverse divisioni surrogate per ogni divisione primaria nell'albero.
È importante capire come andare a gestire i valori nulli all'interno degli alberi decisionali poichè, come

abbiamo visto durante il corso, questa tipologia di modelli sono altamente instabili ovvero se durante la classificazione o regressione cambia il metodo di splitting, questo verrà propagato a tutto l'albero cambiando la variabilità di questo.
Attraverso il `summary` è possibile esaminare i predittori utilizzati nell'albero (compreso i predittori surrogati) e la loro relativa importanza nella previsione, come mostrato di seguito.

```
summary(rpart_mod)
```

```
## Call:
## rpart(formula = Species ~ ., data = scat)
##   n= 110
##
##           CP nsplit rel error    xerror       xstd
## 1 0.26415094      0 1.0000000 1.0000000 0.09887879
## 2 0.16981132      1 0.7358491 0.7169811 0.09409918
## 3 0.13207547      2 0.5660377 0.6792453 0.09285278
## 4 0.03773585      3 0.4339623 0.5660377 0.08813196
## 5 0.01000000      4 0.3962264 0.5094340 0.08516256
##
## Variable importance
##     Mass       CN     d13C     d15N     flat    Month Diameter Location
##       22       22       21       17        5        4        4        2
##     Site
##        2
##
## Node number 1: 110 observations,    complexity param=0.2641509
##   predicted class=bobcat    expected loss=0.4818182  P(node) =1
##     class counts:    57    28    25
##    probabilities: 0.518 0.255 0.227
##   left son=2 (83 obs) right son=3 (27 obs)
##   Primary splits:
##       CN        < 8.7     to the left,  improve=13.135800, (2 missing)
##       d13C      < -24.195 to the left,  improve=12.812480, (2 missing)
##       d15N      < 11.78   to the left,  improve=10.617790, (2 missing)
##       Mass      < 8.64    to the right, improve= 8.667345, (1 missing)
##       segmented < 0.5     to the right, improve= 6.139761, (0 missing)
##   Surrogate splits:
##       flat     < 0.5     to the left,  agree=0.806, adj=0.222, (2 split)
##       Diameter < 11.85   to the right, agree=0.778, adj=0.111, (0 split)
##       Month    splits as  LLLRLLLLL,   agree=0.769, adj=0.074, (0 split)
##       Mass     < 3.625   to the right, agree=0.769, adj=0.074, (0 split)
##
## Node number 2: 83 observations,    complexity param=0.1698113
##   predicted class=bobcat    expected loss=0.3253012  P(node) =0.7545455
##     class counts:    56    17    10
##    probabilities: 0.675 0.205 0.120
##   left son=4 (74 obs) right son=5 (9 obs)
##   Primary splits:
##       d13C     < -24.195 to the left,  improve=11.114200, (2 missing)
##       d15N     < 11.78   to the left,  improve=11.114200, (2 missing)
##       Mass     < 5.44    to the right, improve= 4.761464, (0 missing)
##       Diameter < 22.05   to the left,  improve= 3.034042, (0 missing)
##       CN       < 6.25    to the left,  improve= 3.020382, (2 missing)
##   Surrogate splits:
```

```
##       d15N < 12.44   to the left,  agree=0.963, adj=0.667, (0 split)
##       Mass < 27.5    to the left,  agree=0.901, adj=0.111, (2 split)
##
## Node number 3: 27 observations,    complexity param=0.1320755
##   predicted class=gray_fox  expected loss=0.4444444  P(node) =0.2454545
##     class counts:     1    11    15
##    probabilities: 0.037 0.407 0.556
##   left son=6 (9 obs) right son=7 (18 obs)
##   Primary splits:
##       Mass     < 8.79    to the right, improve=5.520865, (1 missing)
##       Diameter < 17.7    to the right, improve=4.817460, (6 missing)
##       CN       < 15.45   to the left,  improve=4.148148, (0 missing)
##       Month    splits as  RR-LL-LLR,   improve=2.703704, (0 missing)
##       Location splits as  LRL,         improve=1.689325, (0 missing)
##   Surrogate splits:
##       d15N     < 9.795   to the right, agree=0.846, adj=0.556, (1 split)
##       d13C     < -25.43  to the right, agree=0.769, adj=0.333, (0 split)
##       Month    splits as  RR-RR-LRR,   agree=0.731, adj=0.222, (0 split)
##       Site     splits as  RL,          agree=0.731, adj=0.222, (0 split)
##       Location splits as  LRR,         agree=0.731, adj=0.222, (0 split)
##
## Node number 4: 74 observations,    complexity param=0.03773585
##   predicted class=bobcat    expected loss=0.2432432  P(node) =0.6727273
##     class counts:    56     8    10
##    probabilities: 0.757 0.108 0.135
##   left son=8 (58 obs) right son=9 (16 obs)
##   Primary splits:
##       Mass      < 5.44    to the right, improve=5.698509, (0 missing)
##       d15N      < 8.78    to the left,  improve=3.242063, (2 missing)
##       CN        < 6.25    to the left,  improve=2.396971, (2 missing)
##       Month     splits as  RRLRLLRLL,   improve=2.259207, (0 missing)
##       segmented < 0.5     to the right, improve=1.607854, (0 missing)
##   Surrogate splits:
##       Diameter < 12.75   to the right, agree=0.824, adj=0.188, (0 split)
##       Month    splits as  LRLLLLLLL,   agree=0.797, adj=0.062, (0 split)
##
## Node number 5: 9 observations
##   predicted class=coyote    expected loss=0  P(node) =0.08181818
##     class counts:     0     9     0
##    probabilities: 0.000 1.000 0.000
##
## Node number 6: 9 observations
##   predicted class=coyote    expected loss=0.1111111  P(node) =0.08181818
##     class counts:     0     8     1
##    probabilities: 0.000 0.889 0.111
##
## Node number 7: 18 observations
##   predicted class=gray_fox  expected loss=0.2222222  P(node) =0.1636364
##     class counts:     1     3    14
##    probabilities: 0.056 0.167 0.778
##
## Node number 8: 58 observations
##   predicted class=bobcat    expected loss=0.137931  P(node) =0.5272727
##     class counts:    50     6     2
```

```
##    probabilities: 0.862 0.103 0.034
##
## Node number 9: 16 observations
##   predicted class=gray_fox  expected loss=0.5  P(node) =0.1454545
##     class counts:     6     2     8
##    probabilities: 0.375 0.125 0.500
```

Si identifica quindi che la ripartizione iniziale si basa sul rapporto carbonio/azoto (`CN < 8,7`) ma quando un campione ha un valore mancante per la variabile `CN`, il modello CART utilizza una suddivisione alternativa `flat` (indicatore che denota se gli escrementi sono di forma piatta o meno). Scendendo ulteriormente nell'albero, i predittori surrogati per `d13C` e `Mass` sono rispettivamente `Mass` e `d13C`. Ciò è possibile poiché questi predittori non mancano contemporaneamente.
Se si inserisce all'interno del codice il nome dell'oggetto albero, R stampa l'output corrispondente a ciascun ramo. In questo modo R visualizza il criterio di divisione (ad esempio `CN < 8.7`), il numero di osservazioni in quel ramo, la devianza, la previsione complessiva per il ramo (`bobcat`, `gray_fox` o `coyote`) e la frazione di osservazioni in quel ramo che assumono valori di `bobcat`, `gray_fox` o `coyote`. I rami che portano ai nodi terminali sono indicati con asterischi.

```
rpart_mod
```

```
## n= 110
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 110 53 bobcat (0.51818182 0.25454545 0.22727273)
##   2) CN< 8.7 83 27 bobcat (0.67469880 0.20481928 0.12048193)
##     4) d13C< -24.195 74 18 bobcat (0.75675676 0.10810811 0.13513514)
##       8) Mass>=5.44 58  8 bobcat (0.86206897 0.10344828 0.03448276) *
##       9) Mass< 5.44 16  8 gray_fox (0.37500000 0.12500000 0.50000000) *
##     5) d13C>=-24.195 9  0 coyote (0.00000000 1.00000000 0.00000000) *
##   3) CN>=8.7 27 12 gray_fox (0.03703704 0.40740741 0.55555556)
##     6) Mass>=8.79 9  1 coyote (0.00000000 0.88888889 0.11111111) *
##     7) Mass< 8.79 18  4 gray_fox (0.05555556 0.16666667 0.77777778) *
```

Metodi di imputazione

Un altro approccio alla gestione dei valori mancanti consiste nell'imputarli. L'imputazione utilizza informazioni e relazioni tra i predittori non mancanti per fornire una stima per riempire il valore assente.

K-Nearest Neighbors

Quando il training set è di dimensioni ridotte o moderate, *K-nearest neighbors* può essere un metodo rapido ed efficace per imputare valori mancanti. Questa procedura identifica un campione con uno o più valori assenti. Quindi identifica i K campioni più simili nei dati di addestramento che sono completi. Questo metodo viene utilizzato anche per effettuare delle regressioni non parametriche.

In seguito viene utilizzato di nuovo il dataset *scat*.

Per effettuare le analisi conviene evidenziare i valori mancanti, tramite una variabile binaria, delle colonne `diameter` e `mass`.

```r
#Dataset che contiene in più la colonna `was_missing`
scat_missing <-
  scat %>%
  mutate(
    was_missing = ifelse(is.na(Diameter)| is.na(Mass), "yes", "no"),
    was_missing = factor(was_missing, levels = c("yes", "no"))
  )

head(scat_missing[,c("Species", "Diameter", "Mass", "was_missing")], n=15)
```

```
## # A tibble: 15 x 4
##    Species  Diameter  Mass was_missing
##    <fct>       <dbl> <dbl> <fct>
##  1 coyote       25.7 15.9  no
##  2 coyote       25.4 17.6  no
##  3 bobcat       18.8  8.4  no
##  4 coyote       18.1  7.4  no
##  5 coyote       20.7 25.4  no
##  6 coyote       21.2 14.1  no
##  7 bobcat       15.7 14.8  no
##  8 bobcat       21.9 26.4  no
##  9 bobcat       17.5 16.2  no
## 10 bobcat       18   11.2  no
## 11 gray_fox     NA    2.51 yes
## 12 gray_fox     12.9  8.55 no
## 13 gray_fox     NA   18.1  yes
## 14 gray_fox     NA    8.17 yes
## 15 gray_fox     NA    3.43 yes
```

Si mostra l'applicazione dell'algoritmo *K-nearest neighbors* tramite l'inputazione dei *missing values* delle variabili `diameter` e `mass`, usando i valori delle altre variabili. La funzione `step_impute_knn()` del pacchetto `recipes` implementa il metodo usando come valore di default k=5. k è un parametro di tuning, al crescere del suo valore si ha meno flessibilità nel fitting, viceversa a valori bassi di k corrisponde più flessibilità. Al contrario della regressione non parametrica, il vicinato viene calcolato usando la **distanza di Gower** che permette di trattare dati quantitativi e qualitativi.

```r
imp_knn <-
  recipe(Species ~ ., data = scat) %>%
  step_impute_knn(Diameter, Mass,
                  impute_with =
                    imp_vars(Month, Year, Site, Location,
                             Age, Number, Length, ropey,
                             segmented, scrape)) %>%
  prep(training = scat, retain = TRUE) %>%
  juice(Diameter, Mass) %>%
  set_names(c("diam_imp", "mass_imp")) %>%
  mutate(method = "5-Nearest Neighbors")

scat_knn <- bind_cols(scat_missing, imp_knn)
```

Bagged trees

I modelli basati su alberi sono una scelta ragionevole per una tecnica di imputazione poiché un albero può essere costruito in presenza di altri dati mancanti. Inoltre, gli alberi hanno generalmente una buona precisione e non estrapolano valori oltre i limiti dei dati di addestramento. Un singolo albero è noto per produrre risultati che hanno una bassa distorsione ma un'alta varianza. Gli insiemi di alberi, tuttavia, forniscono un'alternativa a bassa varianza. Le *Random Forests* sono una di queste tecniche.

In generale, è sempre utile trovare un *trade-off* tra distorsione e varianza. Un eccesso di varianza può portare all'*overfitting*, viceversa una distorsione molto elevata porta ad un modello più rigido.

Per diminuire il costo computazionale delle *Random Forests* una buona alternativa che ha un ingombro computazionale più piccolo è un *bagged tree*, che è costruito in modo simile a una foresta casuale. La differenza principale è che in un modello di questo tipo, tutti i predittori vengono valutati ad ogni divisione in ogni albero.

Si utilizza quindi anche questo metodo per imputare i valori mancanti di `diameter` e `mass`. Per utilizzare un insieme di 50 *bagged trees* si usa la funzione `bagging()` del pacchetto `ipred`.

```
set.seed(3453)

diam_fit <- bagging(Diameter ~  ., data = scat[, -1],
                    nbagg = 50, coob = TRUE)
#nbagg è il numero di alberi
diam_res <- getModelInfo("treebag")[[1]]$oob(diam_fit)

diam_res
```

```
##       RMSE   Rsquared     RMSESD RsquaredSD
## 4.16092824 0.13614116 0.63086821 0.09929666
```

Quando l'obiettivo è l'imputazione di `diameter`, l'RMSE stimato del modello è 4,16 con un R^2 del 13,6%.

```
set.seed(3453)

mass_fit <- bagging(Mass ~  ., data = scat[, -1],
                    nbagg = 50, coob = TRUE)

mass_res <- getModelInfo("treebag")[[1]]$oob(mass_fit)

mass_res
```

```
##       RMSE   Rsquared     RMSESD RsquaredSD
##  8.5563677  0.2852951  1.7837315  0.1498834
```

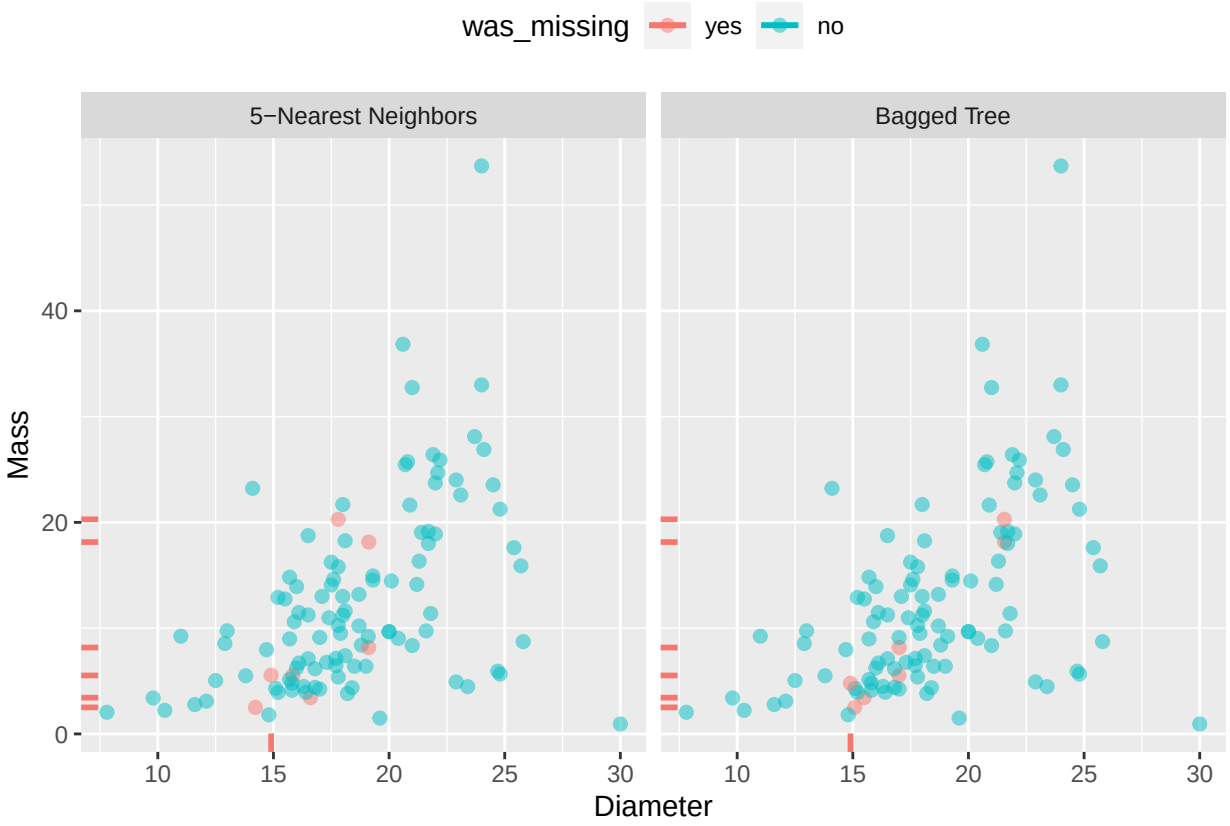
Quando l'obiettivo è l'imputazione di `mass`, l'RMSE stimato del modello è 8,56 con un R^2 del 28,5%.

Tuttavia, queste imputazioni dei *bagged models* producono risultati ragionevoli, rientrano nell'intervallo dei dati di addestramento e consentono di conservare i predittori per la modellazione (al contrario della cancellazione caso per caso).

```
scat_bag <-
  scat_missing %>%
  mutate(method = "Bagged Tree",
         diam_imp = Diameter, mass_imp = Mass)

scat_bag$diam_imp[is.na(scat_bag$Diameter)] <-
  predict(diam_fit, scat[is.na(scat$Diameter),])

scat_bag$mass_imp[is.na(scat_bag$Mass)] <-
  predict(mass_fit, scat[is.na(scat$Mass),])

imputed <- bind_rows(scat_knn, scat_bag)
```

L'immagine seguente mostra i valori imputati ottenuti con *K-nearest neighbors* a sinistra e con i *bagged trees* a destra. Nel primo caso i nuovi valori (in rosa) cadono per lo più intorno alla periferia di queste due dimensioni, ma sono all'interno della gamma dei campioni con dati completi. Nel secondo caso i valori imputati cadono principalmente nelle zone più dense.

```
ggplot(imputed, aes(col = was_missing)) +
  geom_point(aes(x = diam_imp, y = mass_imp), alpha = .5, cex = 2) +
  geom_rug(data = imputed[is.na(imputed$Mass),],
           aes(x = Diameter),
           sides = "b",
           lwd = 1) +
  geom_rug(data = imputed[is.na(imputed$Diameter),],
           aes(y = Mass),
           sides = "l",
           lwd = 1) +
  theme(legend.position = "top") +
  xlab("Diameter") + ylab("Mass") +
  facet_wrap(~method)
```



Linear Methods

Quando un predittore completo mostra una forte relazione lineare con un predittore che richiede l'imputazione, un modello lineare semplice può essere l'approccio migliore. La regressione lineare può essere utilizzata per un predittore numerico che richiede l'imputazione. Allo stesso modo, la regressione logistica è appropriata per un predittore categorico che richiede l'imputazione.

Per esempio, analizzando i dati sull'utenza dei treni di Chicago (usati anche in precedenza) si può notare come il ritardo di 14 giorni nell'utenza all'interno di una fermata è altamente correlato con l'utenza del giorno corrente.

```
# Gestione delle date
train_plot_data <-
  training %>%
  mutate(date = train_days)

train_plot_data <-
  train_plot_data %>%
  mutate( #si inserisce una colonna con un'etichetta riguardo il tipo di giorno
    #della settimana
    pow = ifelse(dow %in% c("Sat", "Sun"), "Weekend", "Weekday"),
    pow = factor(pow)
  )

#Festività
```

```
set.seed(149334)

commonHolidays <-
  c("USNewYearsDay", "Jan02_Mon_Fri", "USMLKingsBirthday",
    "USPresidentsDay", "USMemorialDay", "USIndependenceDay",
    "Jul03_Mon_Fri", "Jul05_Mon_Fri", "USLaborDay", "USThanksgivingDay",
    "Day_after_Thx", "ChristmasEve", "USChristmasDay", "Dec26_wkday",
    "Dec31_Mon_Fri")

any_holiday <- #si inserisce una colonna che etichetta se il giorno è una
  #festività o meno
  train_plot_data %>%
  dplyr::select(date, !!commonHolidays) %>%
  gather(holiday, value, -date) %>%
  group_by(date) %>%
  summarize(common_holiday = max(value)) %>%
  ungroup() %>%
  mutate(common_holiday = ifelse(common_holiday == 1, "Holiday", "Non-holiday")) %>%
  inner_join(train_plot_data, by = "date")

holiday_values <-
  any_holiday %>%
  dplyr::select(date, common_holiday)

#funzione per calcolare il lag di due settimane rispetto ad una data
make_lag <- function(x, lag = 14) {
  x$date <- x$date + days(lag)
  prefix <- ifelse(lag < 10, paste0("0", lag), lag)
  prefix <- paste0("l", prefix, "_holiday")
  names(x) <- gsub("common_holiday", prefix, names(x))
  x
}

#Si laggano i dati di 14 giorni
lag_hol <- make_lag(holiday_values, lag = 14)

holiday_data <- #aggiunta della colonna al dataset totale con i valori delle
  #festività dei giorni laggati
  any_holiday %>%
  left_join(lag_hol, by = "date") %>%
  mutate(
    year = factor(year),
    l14_holiday = ifelse(is.na(l14_holiday), "Non-holiday", l14_holiday)
  )
```

La figura seguente mostra la relazione tra questi predittori per la fermata di Clark/Lake. La maggior parte dei dati mostra una relazione lineare tra questi predittori, con una manciata di giorni che hanno valori lontani dalla tendenza generale. Ovviamente includere le vacanze come predittore nel modello robusto contribuirebbe a migliorare l'imputazione.

```
holiday_data %>%
  dplyr::filter(common_holiday == "Non-holiday" & l14_holiday == "Non-holiday") %>%
  ggplot(aes(l14_40380, s_40380, col = pow)) +
```

```
geom_point(alpha=0.5) +
scale_color_manual(values = c("#D53E4F", "#3ed5c4")) +
xlab("14-Day Lag") +
ylab("Current Day") +
theme(legend.title=element_blank())+
coord_equal() +
geom_abline(linetype = "dashed")
```