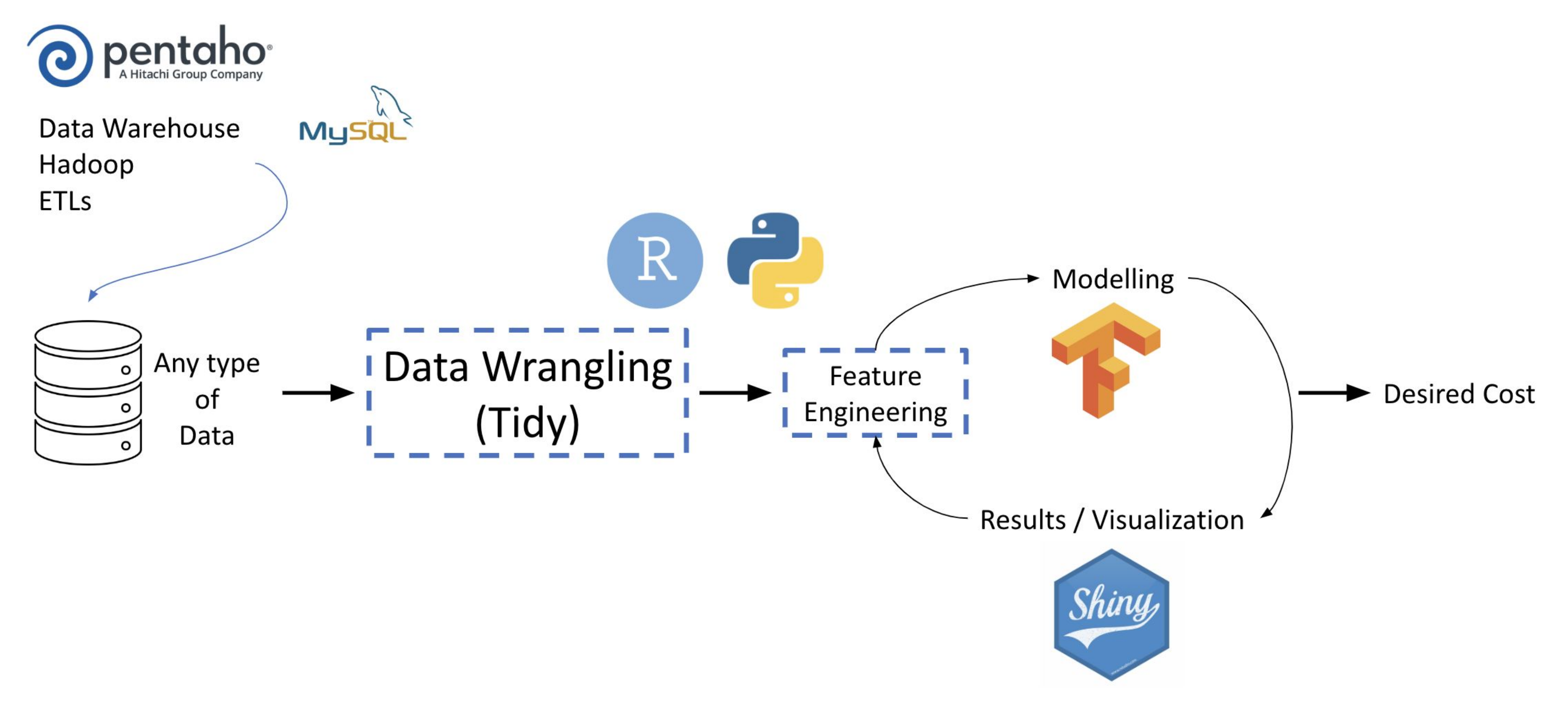
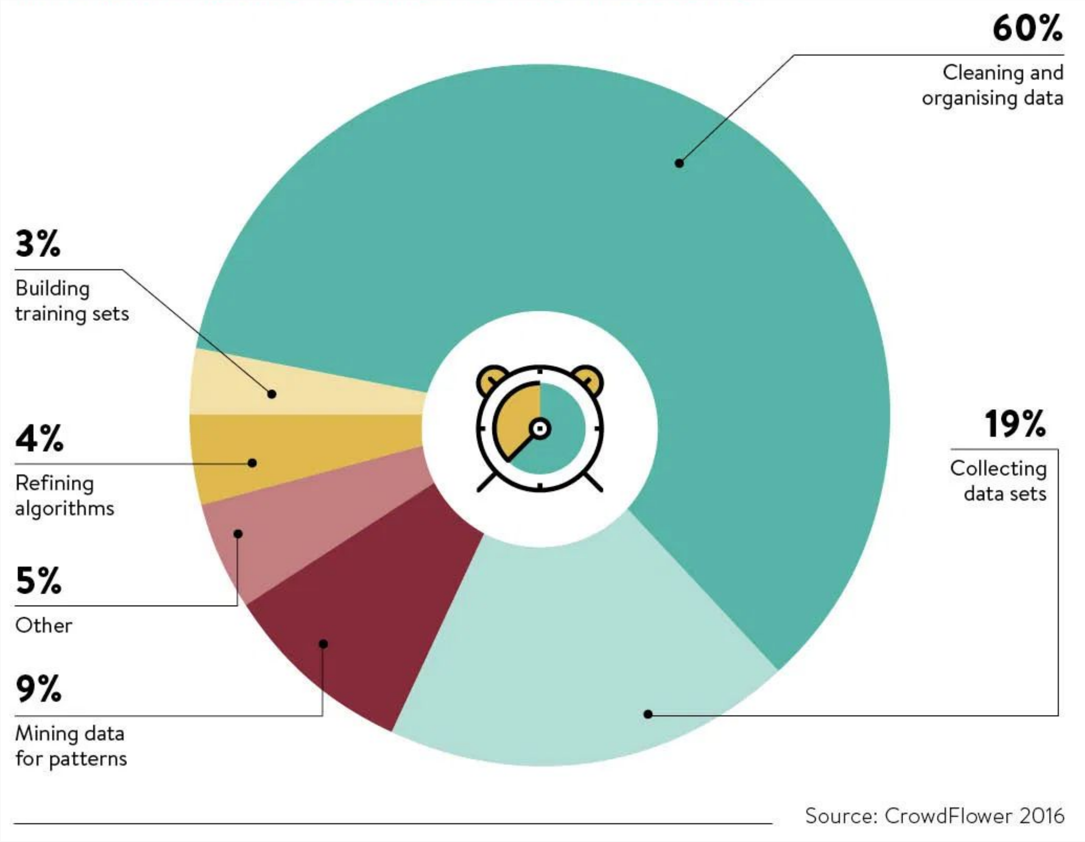
Feature Engineering

Data Wrangling

… from the first session



What data scientists spend the most time doing...



Agenda

1. Missing Data

1.1 Deletion

1.2 Filling and dropping values

1.3 Imputations and Sectorized Imputations

1.4 Prediction models

2. Outliers

3. Normalization (Numerical variables)

3.1 Standardization (Z value)

3.2 Scalers

3.3 Bins

3.4 Log transformations

4. Embeddings (Categorical variables)

4.1 Label encoding

4.2 One hot encoding

4.3 GloVe (Global Vectors)

4.4 Skipgrams and Bag of Words

5. Splitting Sets

5.1 Train, test and validation sets

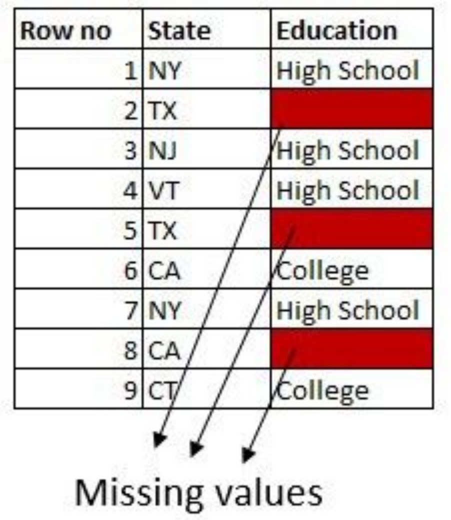
Agenda

1. Missing Data
2. Outliers
3. Normalization (Numerical Variables)
4. Embeddings (Categorical Variables)
5. Splitting Sets

Dealing with Missing Data

- NA
- NULL
- <blank>
- 0, 99, 999

Are NA and <blank> the same?
Can we handle missing data as part of the test set?
Some ML, DL... algorithms drop NA values... some give errors



Row no	State	Education
1	NY	High School
2	TX	
3	NJ	High School
4	VT	High School
5	TX	
6	CA	College
7	NY	High School
8	CA	
9	CT	College


```
> summary(data)
    Fecha              COD_VIAJE          CLIENTE              UBICACION          Tarifa
 Length:2180        Min.   :1e+07   Length:2180        Min.   :76001   Min.   :0.2505
 Class :character   1st Qu.:1e+07   Class :character   1st Qu.:76001   1st Qu.:0.2620
 Mode  :character   Median :1e+07   Mode  :character   Median :76002   Median :0.2738
                    Mean   :1e+07                      Mean   :76002   Mean   :0.2837
                    3rd Qu.:1e+07                      3rd Qu.:76002   3rd Qu.:0.2910
                    Max.   :1e+07                      Max.   :76002   Max.   :0.4748
                                                                       NA's   :169
```

Deletion - Listwise

When to use listwise deletion?

- Data is not entirely available
- Biased sample or results
- You have sufficient data already
- Probability to get a missing value is unrelated (MCAR)
- Missing key features of observation

Advantages: Very easy to implement in any language
Disadvantages: Losing observations

Gender	Manpower	Sales
M	25	343
~~F~~	~~.~~	~~280~~
M	33	332
~~M~~	~~.~~	~~272~~
~~F~~	~~25~~	~~.~~
M	29	326
	~~26~~	~~259~~
M	32	297

```
> complete.cases(data) %>% table()
.
FALSE  TRUE
  452  1728
```

```
> data[complete.cases(data),] %>% head()
     Fecha COD_VIAJE                          CLIENTE UBICACION Tarifa CANTIDAD                   PILOTO      Q CREDITO        UNIDAD C_CLIENTE  MAX Conos Custodio GPS
1 01-01-17  10000540 UNIVERSIDAD FRANCISCO MARROQUIN     76002 0.4270      243          Felipe Villatoro 103.75      90         Panel         2  300     x              x
3 01-01-17  10001428               HOSPITAL ROOSEVELT     76001 0.2901      424        Luis Jaime Urbano 123.00      90         Panel         2  500     x              x
4 01-01-17  10001654              TIENDA LA BENDICION     76001 0.2766     1278  Fernando Mariano Berrio 353.50      60 Camion Grande         3 1300     x        x     x
5 01-01-17  10002112                   POLLO PINULITO     76001 0.2669     1837 Ismael Rodero Monteagudo 490.25      90 Camion Grande         4 1900                    x
8 02-01-17  10001005               HOSPITAL ROOSEVELT     76002 0.2958     1026          Felipe Villatoro 303.50      60 Camion Grande         1 1100     x              x
9 02-01-17  10001788              TAQUERIA EL CHINITO     76001 0.2822     1427       Angel Valdez Alegria 402.75      30 Camion Grande         3 1500     x        x     x
```

Deletion - Pairwise

When to use pairwise deletion?

- Correlation between multiple values
- Want to use all data available
- Don't have much data

Advantages: Not losing observations. Using all data I collected.
Disadvantages: Features with different size and meaning

Gender	Manpower	Sales
M	25	343
F	.	280
M	33	332
M	.	272
F	25	.
M	29	326
	26	259
M	32	297

```
> mean(data$Tarifa)
[1] NA
> mean(data$Tarifa, na.rm = TRUE)
[1] 0.2837334
```

```
> cor(data$Tarifa, data$CANTIDAD, use = "pairwise.complete.obs")
[1] -0.6166118
```

use

an optional character string giving a method for computing covariances in the presence of missing values. This must be (an abbreviation of) one of the strings "everything", "all.obs", "complete.obs", "na.or.complete", or "pairwise.complete.obs".

Press F1 for additional help

Filling and dropping values

Filling Values

- Fill values with information available (time series for example)
- Nearest Neighbor Imputation (NNI) - surveys - distance minimization
- KNN models

Advantages: Not losing observations.
Disadvantages: Missing the target

Dropping Values

- Dropping features lower than a threshold (70% for example)

Advantages: Not using features with many NA. Can alter our model.
Disadvantages: Losing information

```
> is.na(data_fill$Year) %>% table() %>% prop.table()
.
     FALSE       TRUE
0.08333333 0.91666667
```

```
> data_fill
   Month Year
1      1 2000
2      2   NA
3      3   NA
4      4   NA
5      5   NA
6      6   NA
7      7   NA
8      8 2001
9      9   NA
10    10   NA
11    11   NA
12    12   NA
> fill(data_fill, Year)
   Month Year
1      1 2000
2      2 2000
3      3 2000
4      4 2000
5      5 2000
6      6 2000
7      7 2000
8      8 2001
9      9 2001
10    10 2001
11    11 2001
12    12 2001
```

Imputations - mean/mode/median and 0

- Mean: maintains the average of each feature
- Mode: most repeated value (stats change)
- Median: median value (stats change)
- Constant: Adding a constant to all missing values (stats changes but functions become available without na.rm)

Original

```
> summary(data$Tarifa)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
 0.2505  0.2620  0.2738  0.2837  0.2910  0.4748     169
```

Mean

```
> data$Tarifa[is.na(data$Tarifa)] <- mean(data$Tarifa, na.rm = TRUE)
> summary(data$Tarifa)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2505  0.2627  0.2755  0.2837  0.2886  0.4748
```

Median

```
> data$Tarifa[is.na(data$Tarifa)] <- median(data$Tarifa, na.rm = TRUE)
> summary(data$Tarifa)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2505  0.2627  0.2738  0.2830  0.2886  0.4748
```

Mode

```
> data$Tarifa[is.na(data$Tarifa)] <- find.mode(data$Tarifa)
> summary(data$Tarifa)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.2505  0.2596  0.2717  0.2819  0.2886  0.4748
```

Zero

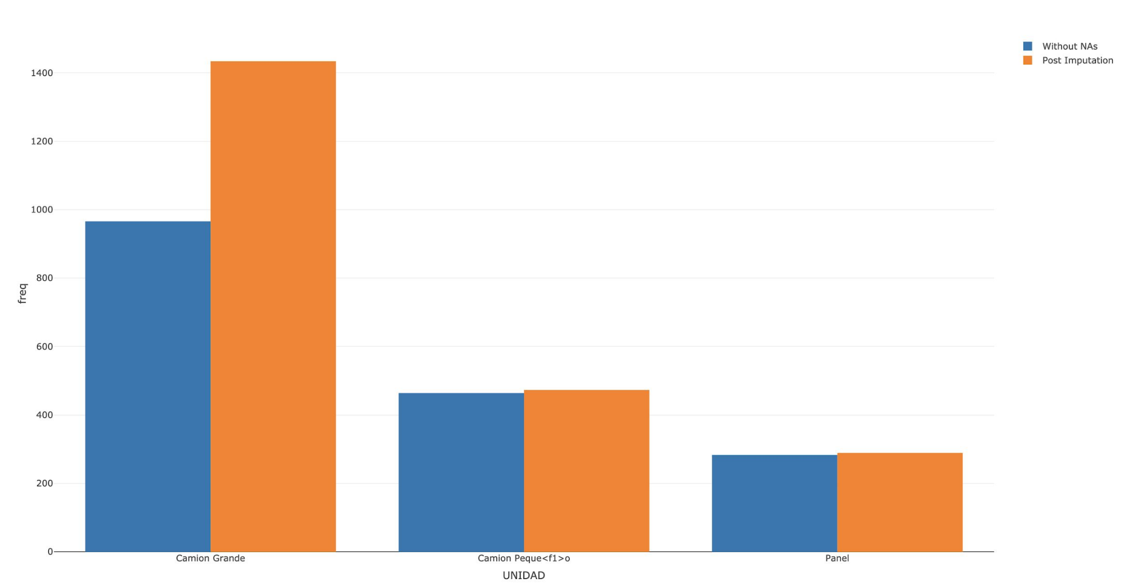
```
> data$Tarifa[is.na(data$Tarifa)] <- 0
> summary(data$Tarifa)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.0000  0.2593  0.2717  0.2617  0.2886  0.4748
```

Categorical and Sectorized Imputations

```
> dic <- find.mode.cat(data, "CLIENTE", "UNIDAD", TRUE)
> data %>%
+   dplyr::filter(!is.na(UNIDAD)) %>%
+   dplyr::group_by(UNIDAD) %>%
+   dplyr::summarise(freq=n()) %>%
+   dplyr::left_join(data %>% left_join(dic, by=c("CLIENTE")) %>% mutate(UNIDAD_NEW=ifelse(is.na(UNIDAD.x), UNIDAD.y, UNIDAD.x)) %>% dplyr::group_
by(UNIDAD = UNIDAD_NEW) %>% summarise(freq1=n()), by = "UNIDAD") %>%
+   plotly::plot_ly(x=~UNIDAD, y=~freq, type = "bar", name = "Without NAs") %>%
+   plotly::add_trace(y=~freq1, name = "Post Imputation")
```

- Using mean/mode/median or 0 imputation by feature.
- Mode is needed for categorical features

Disadvantage: Data is now highly biased



Predictive models

Predictive models are used to choose the best approximation for a missing value.

Numerical features:

- Linear regression
- Multidimensional regression

Categorical features:

- K-Nearest neighbors (KNN)
- Support vector machines (SVM)
- Logistic regression

$$\hat{y} = \beta_0 + \beta_1 x + \epsilon$$

```
def age_prediction(x,y):
    lm = LinearRegression()
    lm.fit(X=x, y=y)
    y_hat = lm.predict(x)
    error = (1/2*np.mean((y_hat-y)**2))

    #building df
    df_dict = {'Id': X.dropna().Id, 'y_hat': y_hat}
    df = pd.DataFrame(df_dict)
    return(lm.coef_, lm.intercept_, error, lm)
```

```
age_lm = age_prediction(X.dropna().drop(['Id', 'Embarked', 'Class', 'Sex', 'Age'], axis = 1), X[
'Age'].dropna())
```

Now that we have the linear model for the prediction of age in place, we'll predict the new values for **Age** in a new column called ***Age_lm***

```
X['age_lm'] = age_lm[3].predict(X.drop(['Id', 'Embarked', 'Class', 'Sex', 'Age'], axis = 1))
X['Age_LM'] = np.where(X['Age']>0, X['Age'], np.where(X['age_lm']>0, X['age_lm'], 1))
X = X.drop('age_lm', axis = 1)
X.head()
```

	Id	Age	SibSp	ParCh	Fare	Embarked	Class	Sex	embarked_code	class_code	sex_code	Age_LM
0	1	22.0	1	0	7.2500	S	Lower	M	2	0	1	22.0
1	2	38.0	1	0	71.2833	C	Upper	F	0	2	0	38.0
2	3	26.0	0	0	7.9250	S	Lower	F	2	0	0	26.0
3	4	35.0	1	0	53.1000	S	Upper	F	2	2	0	35.0
4	5	35.0	0	0	8.0500	S	Lower	M	2	0	1	35.0

Agenda

1. Missing Data
2. Outliers
3. Normalization (Numerical Variables)
4. Embeddings (Categorical Variables)
5. Splitting Sets

Outliers - Standard deviation approach

All values with a distance to the average of **factor * standard deviation** are assumed to be outliers.

How we choose a **factor**? (2-4 recommended)
*We can also use Z value

We can drop the Outliers from our data or we can "Cap" those values. For the observations with a value under the lower limit, we use the lower limit instead; the same for the upper values.

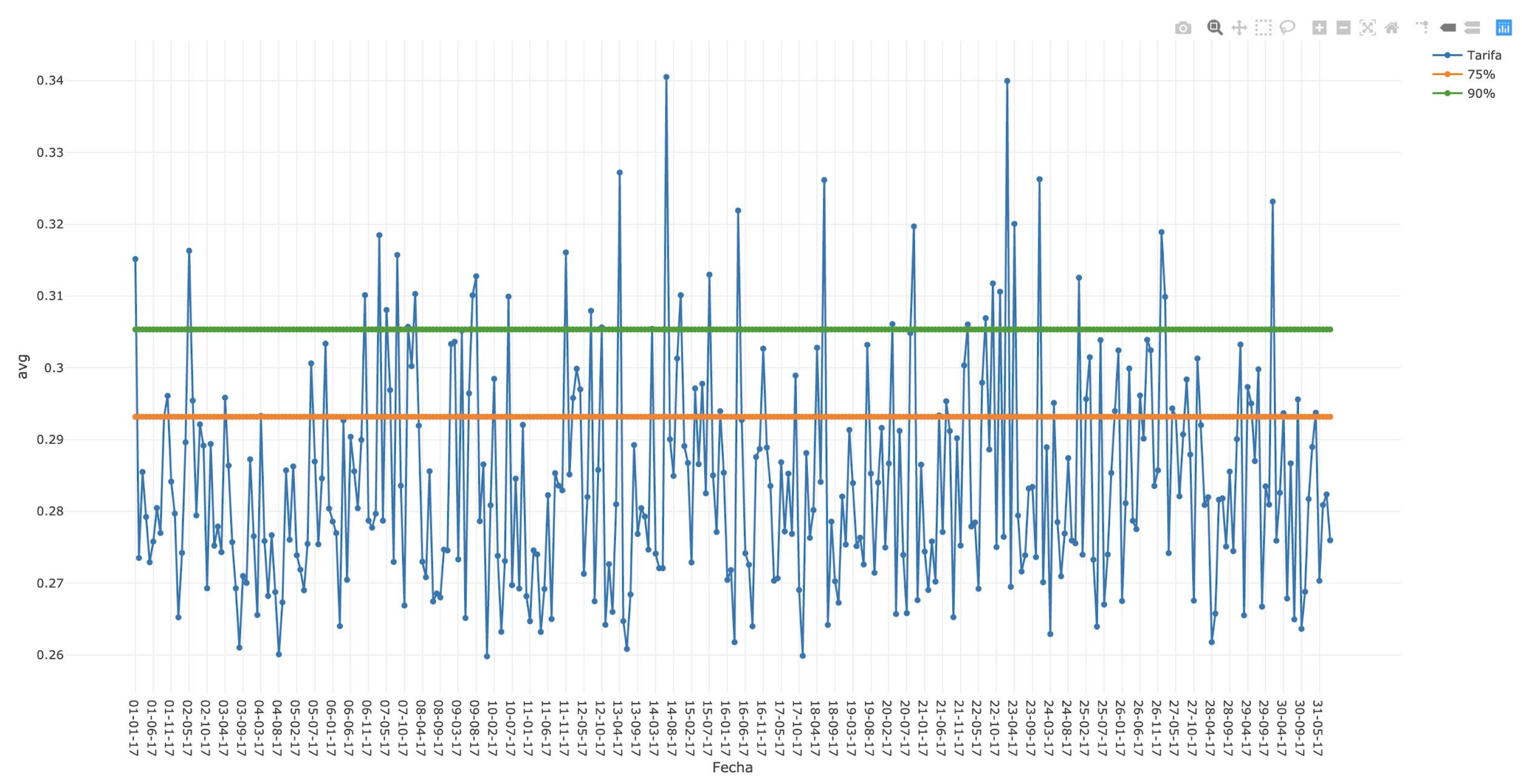
$$x_l = \mu_{var} - \sigma_{var} * x$$

$$x_u = \mu_{var} + \sigma_{var} * x$$

```
> x <- 3
> lower <- mean(data$Tarifa, na.rm = T) - (sd(data$Tarifa, na.rm = T) * x)
> upper <- mean(data$Tarifa, na.rm = T) + (sd(data$Tarifa, na.rm = T) * x)
> lower
[1] 0.1803729
> upper
[1] 0.3870939
```

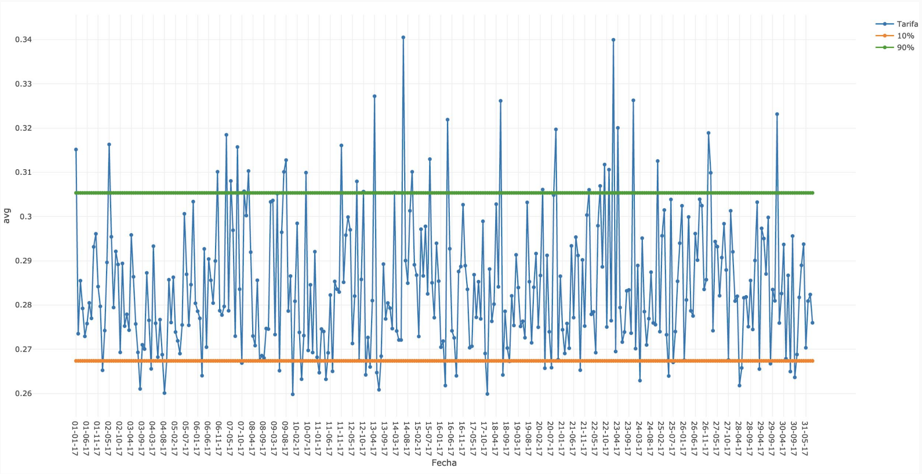
```
> data[ (data$Tarifa>=lower) & (data$Tarifa<=upper) & (!is.na(data$Tarifa)), ]
      Fecha COD_VIAJE                CLIENTE UBICACION Tarifa CANTIDAD                    PILOTO      Q CREDITO
3  01-01-17  10001428      HOSPITAL ROOSEVELT     76001 0.2901      424        Luis Jaime Urbano 123.00      90
4  01-01-17  10001654    TIENDA LA BENDICION     76001 0.2766     1278  Fernando Mariano Berrio 353.50      60
5  01-01-17  10002112         POLLO PINULITO     76001 0.2669     1837  Ismael Rodero Monteagudo 490.25      90
7  02-01-17  10000730            UBIQUO LABS     76001 0.2611      718    Hector Aragones Frutos 187.50      30
8  02-01-17  10001005      HOSPITAL ROOSEVELT     76002 0.2958     1026        Felipe Villatoro 303.50      60
```

Outliers - Percentile approach



Outliers - Percentile approach

```
> p_lower <- quantile(data$Tarifa, na.rm = TRUE, probs = 0.10)
> p_upper <- quantile(data$Tarifa, na.rm = TRUE, probs = 0.90)
> p_lower
  10%
0.2552
> p_upper
  90%
0.3246
```

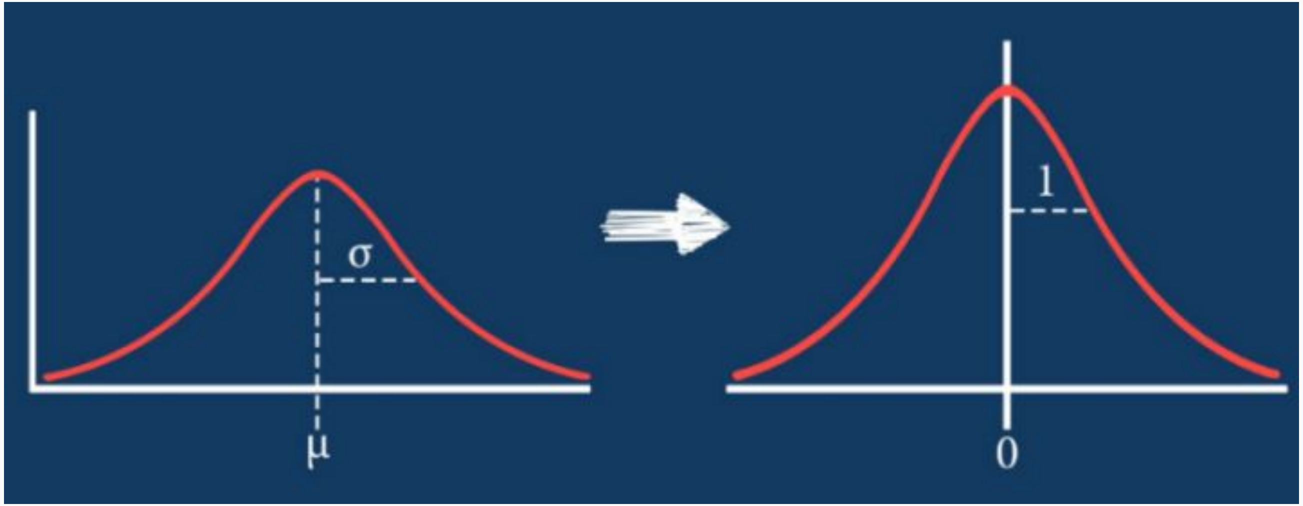



```
> data[ (data$Tarifa>=p_lower) & (data$Tarifa<=p_upper) & (!is.na(data$Tarifa)), ]
     Fecha COD_VIAJE                  CLIENTE UBICACION Tarifa CANTIDAD                         PILOTO      Q CREDITO
3 01-01-17  10001428       HOSPITAL ROOSEVELT     76001 0.2901      424            Luis Jaime Urbano 123.00      90
4 01-01-17  10001654     TIENDA LA BENDICION     76001 0.2766     1278      Fernando Mariano Berrio 353.50      60
5 01-01-17  10002112          POLLO PINULITO     76001 0.2669     1837      Ismael Rodero Monteagudo 490.25      90
7 02-01-17  10000730             UBIQUO LABS     76001 0.2611      718        Hector Aragones Frutos 187.50      30
8 02-01-17  10001005      HOSPITAL ROOSEVELT     76002 0.2958     1026            Felipe Villatoro 303.50      60
9 02-01-17  10001788     TAQUERIA EL CHINITO     76001 0.2822     1427          Angel Valdez Alegria 402.75      30
```

Agenda

1. Missing Data
2. Outliers
3. Normalization (Numerical Variables)
4. Embeddings (Categorical Variables)
5. Splitting Sets

Standardization (Z value)

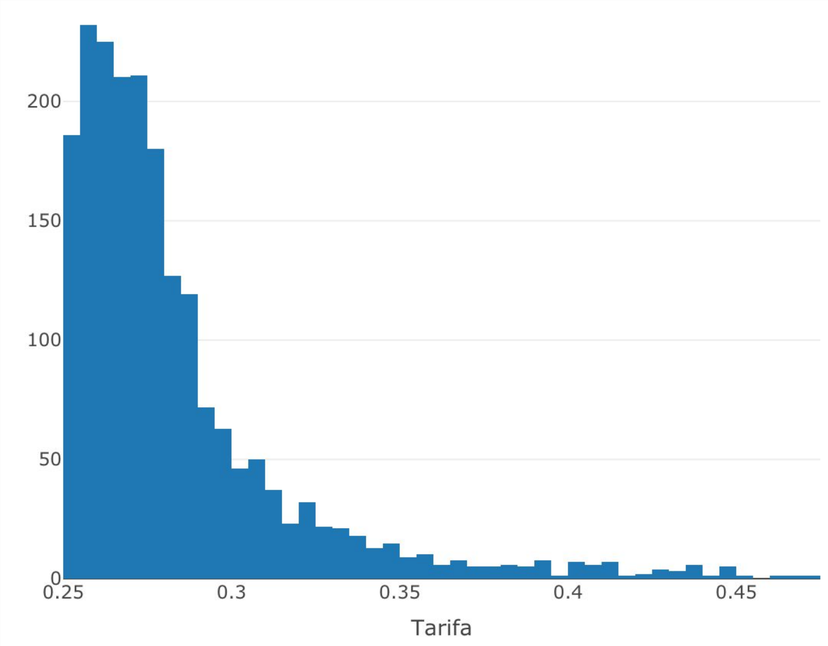


$$z = \frac{x - \mu}{\sigma}$$

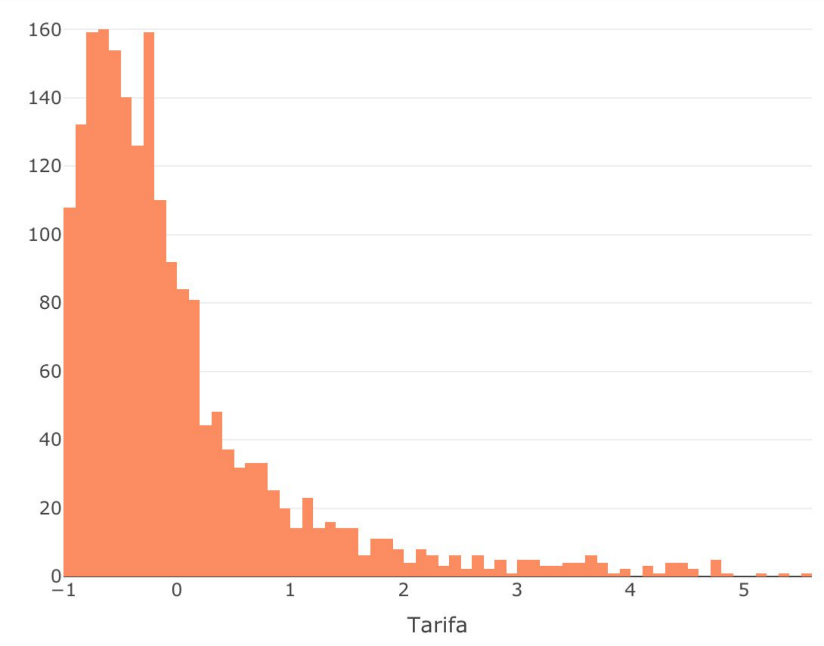
```
> data %>%
+   mutate(Tarifa_NEW = ((Tarifa-mean(Tarifa, na.rm=TRUE))/sd(Tarifa, na.rm = TRUE))) %>%
+   select(Tarifa, Tarifa_NEW) %>%
+   head()
  Tarifa Tarifa_NEW
1 0.4270  4.1582580
2     NA         NA
3 0.2901  0.1847877
4 0.2766 -0.2070447
5 0.2669 -0.4885835
6     NA         NA
```

Standardization (Z value)

Original Histogram



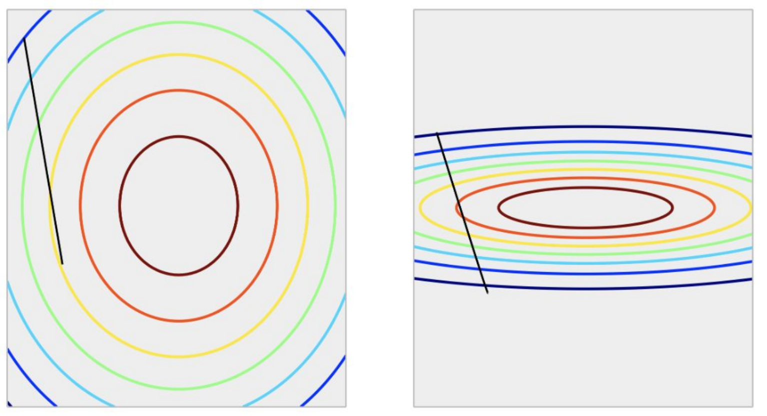
Standardized Histogram



Min Max Scaling (Normalization)

Rescales feature values from 0 to 1

$$x_i^r = \frac{x_i - \min x}{\max x - \min x}$$

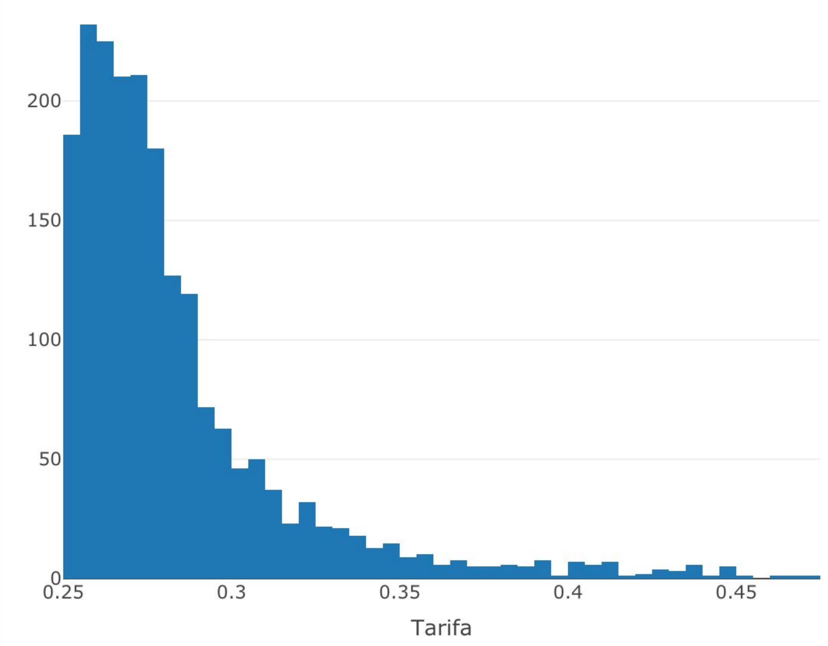


Poorly scaled function

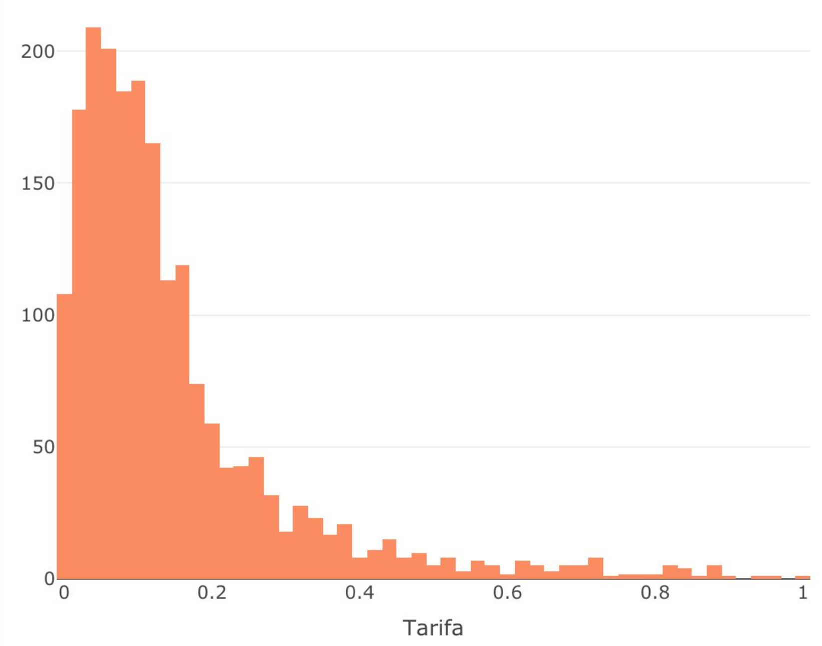
```
> data %>%
+   mutate(Tarifa_NEW = ((Tarifa-min(Tarifa, na.rm=TRUE))/(max(Tarifa, na.rm = TRUE)-min(Tarifa, na.rm = TRUE)))) %>%
+   select(Tarifa, Tarifa_NEW) %>%
+   head()
  Tarifa Tarifa_NEW
1 0.4270 0.78689255
2     NA         NA
3 0.2901 0.17654926
4 0.2766 0.11636202
5 0.2669 0.07311636
6     NA         NA
```

Min Max Scaling (Normalization)

Original Histogram



Normalized Histogram



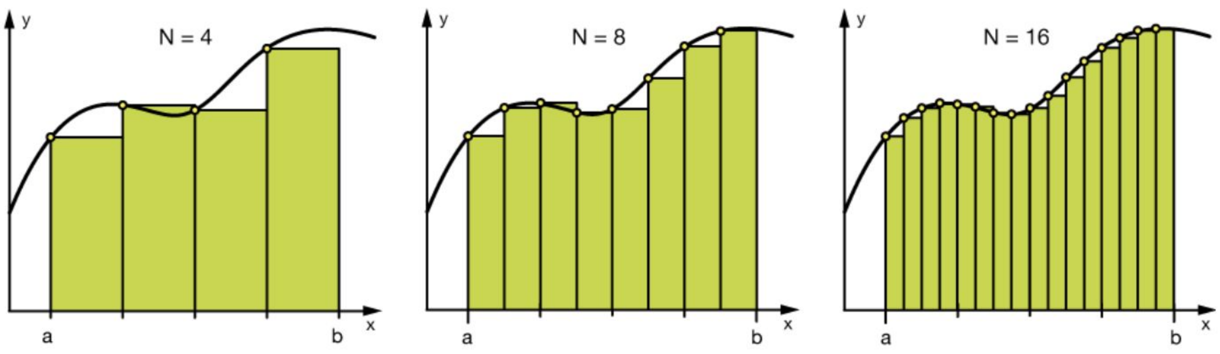
Bins

```
#Numerical Binning Example

Value      Bin
0-30   ->  Low
31-70  ->  Mid
71-100 ->  High

#Categorical Binning Example

Value      Bin
Spain  ->  Europe
Italy  ->  Europe
Chile  ->  South America
Brazil ->  South America
```



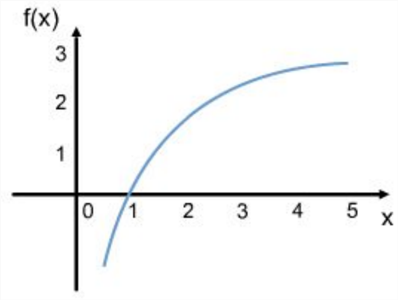
Binning illustration of numerical data

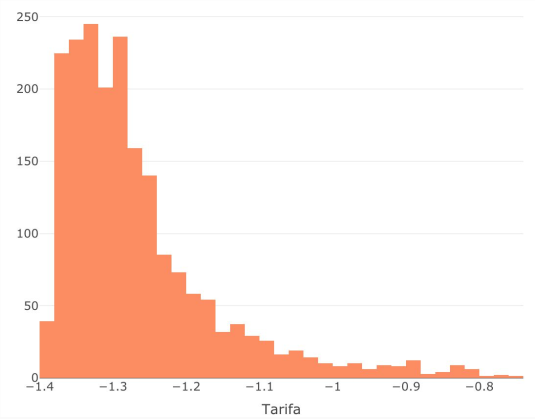
```
> data %>%
+   mutate(CANTIDAD_CAT = ifelse(CANTIDAD <= low,
+                               "Low",
+                               ifelse(CANTIDAD <= med,
+                                      "Med",
+                                      "High")
+                               )) %>% select(CANTIDAD, CANTIDAD_CAT) %>% head()
  CANTIDAD CANTIDAD_CAT
1      243          Low
2     1147         High
3      424          Med
4     1278         High
5     1837         High
6      596          Med
```

- Helps preventing overfitting
- Adds additional features to the model to understand the data
- Splitting into columns

Log Transformations

- Logarithmic transformation (only positive values)
- Helps with skew features
- Features closer to normal distribution
- Some differences in a feature are not linear; age for example.






```
> data %>%
+   mutate(Tarifa_NEW = log(Tarifa)) %>%
+   select(Tarifa, Tarifa_NEW) %>%
+   head()
  Tarifa Tarifa_NEW
1 0.4270 -0.8509713
2     NA         NA
3 0.2901 -1.2375296
4 0.2766 -1.2851829
5 0.2669 -1.3208812
6     NA         NA
```

`log(x, base = exp(1))`

log computes logarithms, by default natural logarithms, log10 computes common (i.e., base 10) logarithms, and log2 computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base *base*.

`log1p(x)` computes *log(1+x)* accurately also for *|x| << 1*.

exp computes the exponential function.

Press F1 for additional help

Log transformation

Agenda

1. Missing Data
2. Outliers
3. Normalization (Numerical Variables)
4. Embeddings (Categorical Variables)
5. Splitting Sets

Label Encoding

- Label encoders allow us to transform a categorical variable to a numerical variable.
- Some frameworks only allow numeric variables (Tensorflow, Keras, Pytorch)
- Useful for transforming ***y*** into classification models.

```
> data$PILOTO_NEW <- as.integer(factor(data$PILOTO))
> data %>% select(PILOTO, PILOTO_NEW) %>% head()
                    PILOTO PILOTO_NEW
1         Felipe Villatoro          2
2   Hector Aragones Frutos          4
3         Luis Jaime Urbano          8
4  Fernando Mariano Berrio          3
5 Ismael Rodero Monteagudo          6
6         Luis Jaime Urbano          8
```

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

CatEncoders

```
> CatEncoders::LabelEncoder.fit(data$UBICACION)
An object of class "LabelEncoder.Numeric"
Slot "classes":
[1] 76001 76002

Slot "type":
[1] "numeric"

Slot "mapping":
  classes ind
1   76001   1
2   76002   2
```

```
> encoder <- LabelEncoder.fit(data$UBICACION)
> data$UBICACION_NEW <- transform(encoder, data$UBICACION)
> data %>% select(UBICACION, UBICACION_NEW) %>% head()
  UBICACION UBICACION_NEW
1     76002             2
2     76001             1
3     76001             1
4     76001             1
5     76001             1
6     76002             2
```

One Hot Encoding

Used to transform a categorical variable into as many binary vectors as categories (levels)

Why *"One Hot"*?

- Dummy variables are used to fit models, allowing to assume a natural ordering between categories. Some of them are present and some aren't.
- Giving a computer categorical variables is talking in a different language that the machine doesn't understand.

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

caret

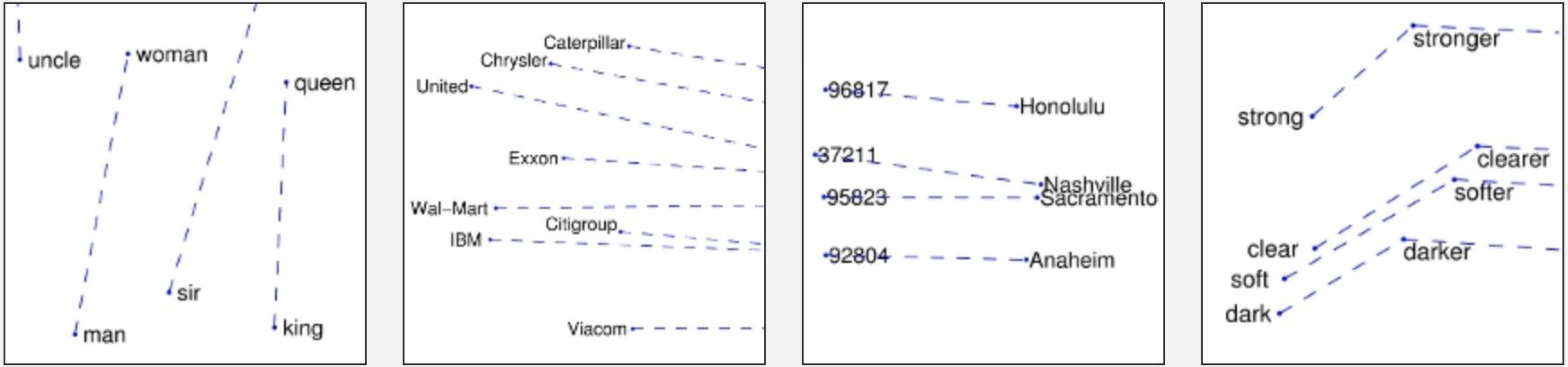
```
> dummies <- dummyVars(Fecha~., data = forOneHot)
> head(predict(dummies, newdata = forOneHot))
  COD_VIAJE UBICACION.76001 UBICACION.76002
1  10000540               0               1
2  10001010               1               0
3  10001428               1               0
4  10001654               1               0
5  10002112               1               0
6  10000046               0               1
```

```
> forOneHot <- data %>% select(Fecha, COD_VIAJE, UBICACION) %>% mutate(val = 1)
> forOneHot %>% spread(key = UBICACION, value = val, fill = 0) %>% head()
     Fecha COD_VIAJE 76001 76002
1 01-01-17  10000540     0     1
2 01-01-17  10001010     1     0
3 01-01-17  10001428     1     0
4 01-01-17  10001654     1     0
5 01-01-17  10002112     1     0
6 01-02-17  10000521     1     0
```

GloVe (Global Vectors)

Ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning which can be encoded as vector differences.

Pre-trained library; GloVe is designed in order that such vector differences capture as much as possible the meaning specified by the juxtaposition of two words.

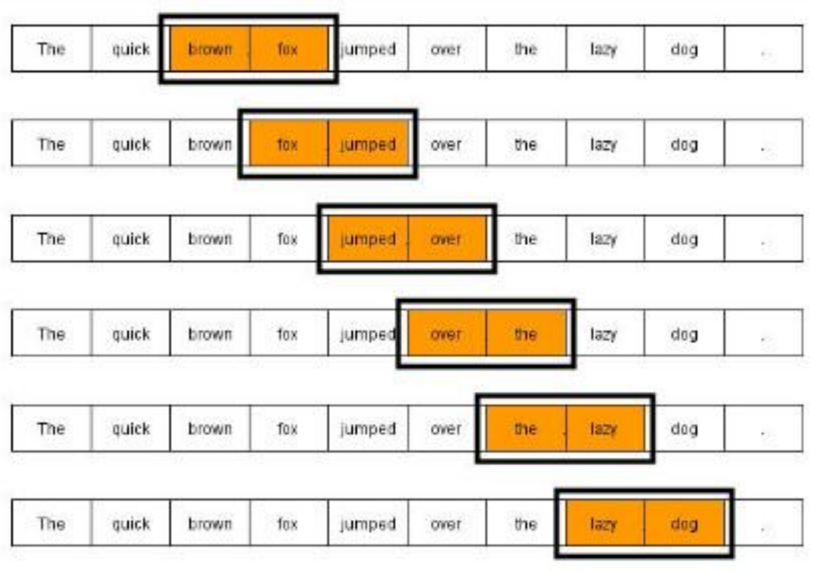


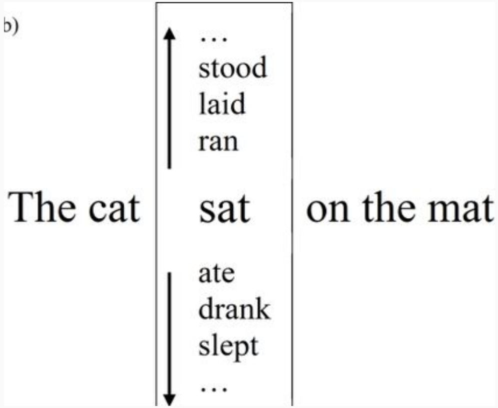
man - woman   company - ceo   city - zip code   comparative - superlative

Skipgrams and Bag of Words (n-grams)

The ‘n’ in n-grams is usually considered a number. So, for example, a 2-gram search will find all 2-item contiguous sequences in a corpus, while a 5-gram search will find all 5-item contiguous sequences in a corpus.

Whereas an n-gram search can only find patterns such as [A+B], skipgrams can find both [A+B] and [A+C+B]. That is, they can find collocates even when there is variety in constituency. A skipgram would uncover “dirty money in politics” and “dirty money in our politics”





Agenda

1. Missing Data
2. Outliers
3. Normalization (Numerical Variables)
4. Embeddings (Categorical Variables)
5. Splitting Sets

Train, Test, and Validation Sets

caret

```
forTrain <- createDataPartition(data$CLIENTE, p = 0.8, list = FALSE)
train <- data[forTrain,]
test <- data[-forTrain,]

forVal <- createDataPartition(test$CLIENTE, p = 0.5, list = FALSE)
val <- test[forVal,]
test <- test[-forVal,]
```

Train set is used to "train" the model (homework, labs, exercises, study)

Validation set is used to provide an unbiased validation (short examinations)

Test set is used to provide unbiased evaluation of a final model already trained (final exam)

*sample function in R

```
"Shape of original data: (2180,17)"
```

~80%

```
"Shape of training set: (1751,17)"
```

~20%

```
"Shape of validation set: (221,17)"
```

```
"Shape of test set: (208,17)"
```