

IMD0033 - Probabilidade

Aula 14 - Distribuições de Frequência

Ivanovitch Silva
Abril, 2019



Agenda

- Tabelas de distribuição de frequências
- Ordenação de tabelas de distribuição de frequências (nominal, ordinal, intervalar, razão)
- Proporções e porcentagens
- Percentil e classificação de percentis
- Agrupamento de tabelas de distribuição de frequência
- Perda de informação

Atualizar o repositório

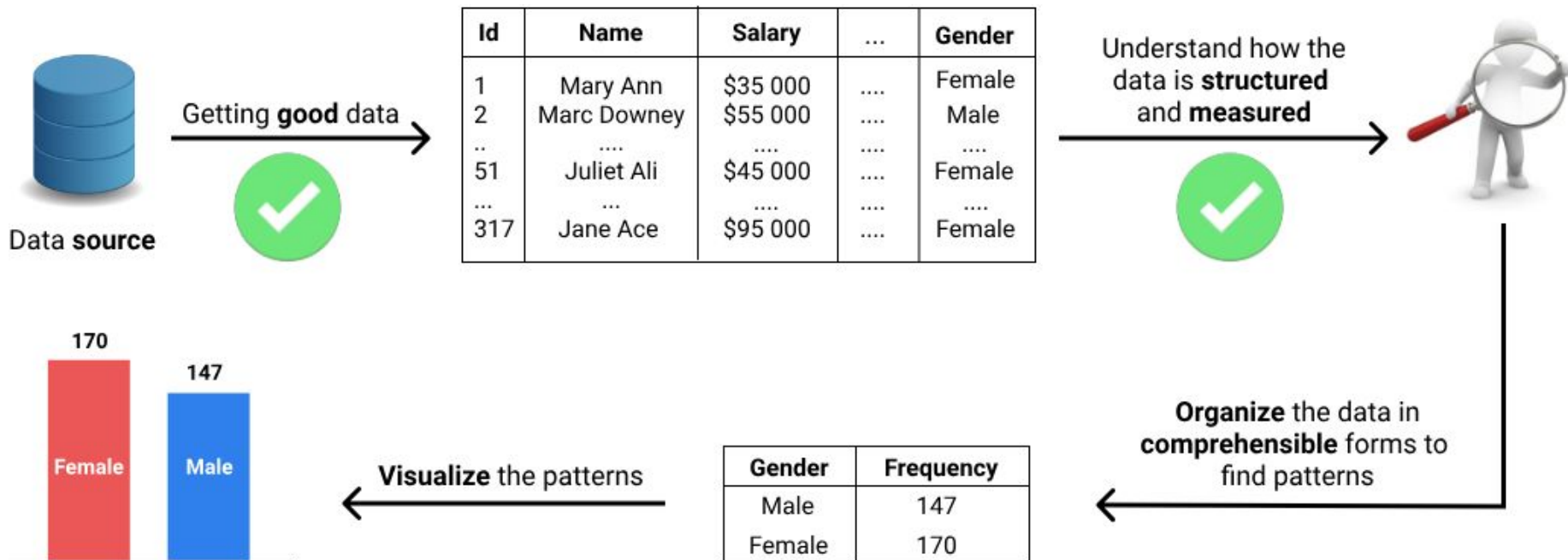
```
git clone https://github.com/ivanovitchm/imd0033_2019_1.git
```

Ou

```
git pull
```

PREVIOUSLY ON...

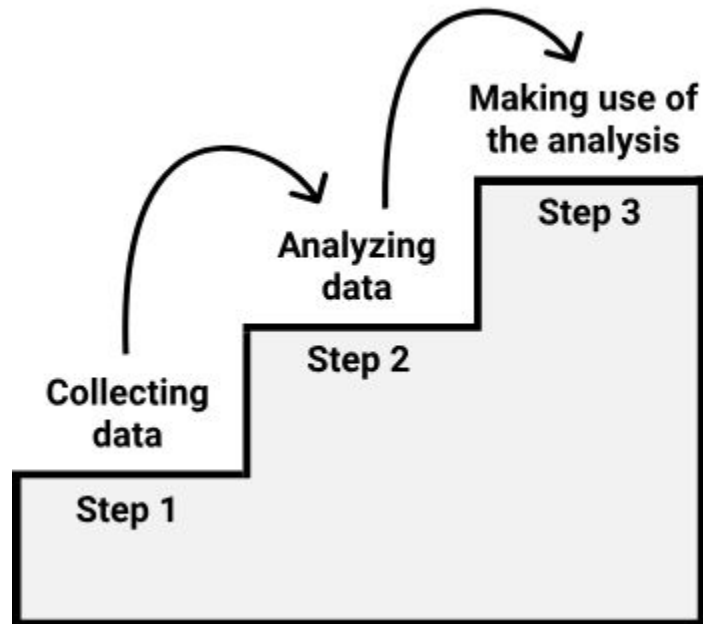
(1) collecting data (2) understanding its structure and how it's measured



Simplifying Data

We collect data to analyze it, and we analyze it for different purposes:

- To describe phenomena in the world (science).
- To make better decisions (industries).
- To improve systems (engineering).
- To describe different aspects of our society (data journalism); etc.



Frequency Distribution Tables

Our capacity to understand a data set just by looking at it in a table format is limited

Position		Frequency
Unique values	G	60
	F	33
	C	25
	G/F	13
	F/C	12
		Frequencies

_	Name	Team	Pos	Height	Weight	BMI	Birth_Place	Birthdate	Age	College	Experience	Games Played	MIN
0	Aerial Powers	DAL	F	183	71.0	21.200991	US	January 17, 1994	23	Michigan State	2	8	173
1	Alana Beard	LA	G/F	185	73.0	21.329438	US	May 14, 1982	35	Duke	12	30	947
2	Alex Bentley	CON	G	170	69.0	23.875433	US	October 27, 1990	26	Penn State	4	26	617
3	Alex Montgomery	SAN	G/F	185	84.0	24.543462	US	December 11, 1988	28	Georgia Tech	6	31	721
4	Alexis Jones	MIN	G	175	78.0	25.469388	US	August 5, 1994	23	Baylor	R	24	137

Sorting Frequency Distribution Tables

```
1 wnba.Pos.value_counts(.)
```

G	60
F	33
C	25
G/F	13
F/C	12

Name: Pos, dtype: int64

Nominal Scale

```
1 wnba.Height.value_counts(.)
```

188	20
193	18
175	16
185	15
191	11
183	11
173	11
196	9
178	8
180	7
170	6
198	5
201	2
168	2
206	1
165	1

How many players are under 170 cm?

How many players are very tall (over 185)?

Are there any players below 160 cm?

Ratio Scale

We can tell the direction of difference

Name: Height, dtype: int64

Sorting Tables for Ordinal Variables

The sorting techniques learned in the previous screen can't be used for ordinal scales where the measurement is done using words.

Condition	Label
points \leq 20	very few points
20 < points \leq 80	few points
80 < points \leq 150	many points
points > 150	a lot of points

	Name	PTS	PTS_ordinal_scale
0	Aerial Powers	93	many points
1	Alana Beard	217	a lot of points
2	Alex Bentley	218	a lot of points
3	Alex Montgomery	188	a lot of points
4	Alexis Jones	50	few points

Sorting Tables for Ordinal Variables

```
>> wnba['PTS_ordinal_scale'].value_counts()
```

```
a lot of points      79
```

```
few points          27
```

```
many points         25
```

```
very few points     12
```

```
dtype: int64
```

```
>> wnba['PTS_ordinal_scale'].value_counts().sort_index()
```

```
very few points     12
```

```
few points          27
```

```
many points         25
```

```
a lot of points      79
```

```
dtype: int64
```

Sorting Tables for Ordinal Variables

```
>> wnba['PTS_ordinal_scale'].value_counts()
```

```
a lot of points      79
```

```
few points          27
```

```
many points         25
```

```
very few points     12
```

```
dtype: int64
```

```
>> wnba['PTS_ordinal_scale'].value_counts().iloc[[3, 1, 2, 0]]
```

```
very few points     12
```

```
few points          27
```

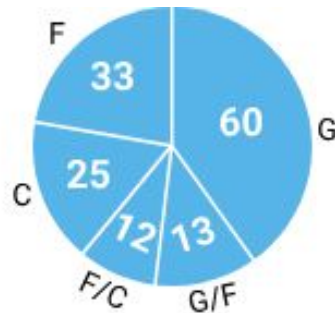
```
many points         25
```

```
a lot of points     79
```

```
dtype: int64
```

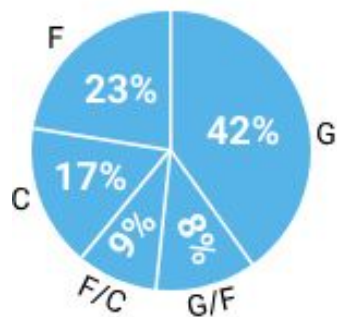
143 players

Counting players
by position

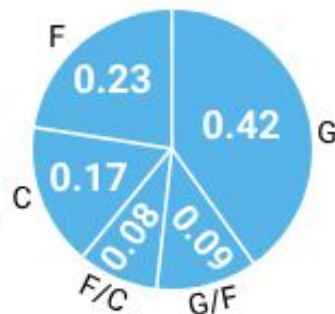


Divide frequencies by
143 to get **proportions***

```
>> wnba['Pos'].value_counts(normalize = True) * 100  
G      41.958042  
F      23.076923  
C      17.482517  
G/F     9.090909  
F/C     8.391608
```



Multiply proportions
by 100 to get **percentages**

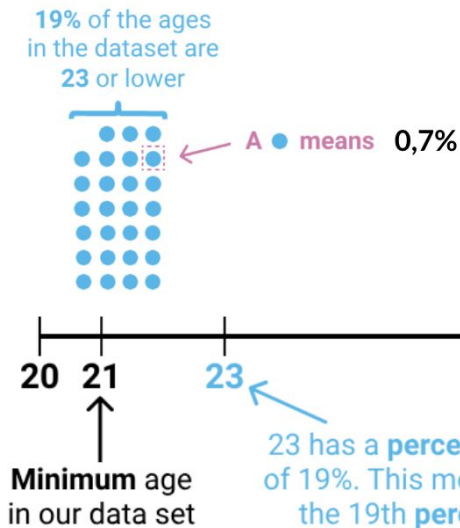


Proportions and Percentages

Percentiles and Percentile Ranks

```
1 percentages = wnba['Age'].value_counts(normalize = True).sort_index() * 100
2 percentages
```

```
21    1.398601
22    6.993007
23   10.489510
24   11.188811
25   10.489510
26    8.391608
27    9.090909
28    9.790210
29    5.594406
30    6.293706
31    5.594406
32    5.594406
33    2.097902
34    3.496503
35    2.797203
36    0.699301
```



"What percentage of players are 23 years or younger?"

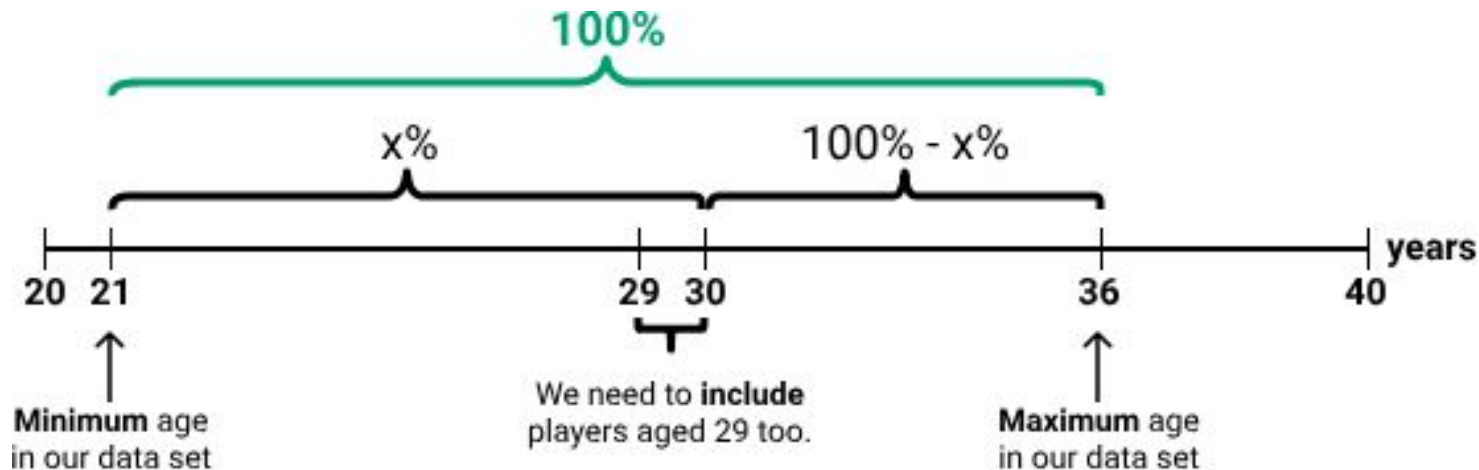
Percentiles and Percentile Ranks

"What percentage of players are 23 years or younger?"

```
>> from scipy.stats import percentileofscore  
>> percentileofscore(a = wnba['Age'], score = 23, kind = 'weak')  
18.88111888111888
```

Percentiles and Percentile Ranks

"What percentage of players are 30 years or older?"



```
>> 100 - percentileofscore(wnba['Age'], 29, kind = 'weak')  
26.573426573426573
```

Finding Percentiles with Pandas

```
>> wnba['Age'].describe()
```

```
count      143.000000
```

```
mean        27.076923
```

```
std          3.679170
```

```
min          21.000000
```

```
25%          24.000000
```

```
50%          27.000000
```

```
75%          30.000000
```

```
max          36.000000
```

```
Name: Age, dtype: float64
```

```
>>wnba['Age'].describe().iloc[3:]
```

```
min      21.0
```

```
25%      24.0
```

```
50%      27.0
```

```
75%      30.0
```

```
max      36.0
```

```
Name: Age, dtype: float64
```

Finding Percentiles with Pandas

```
>> wnba['Age'].describe(percentiles = [.1, .15, .33, .5, .592, .85, .9]).iloc[3:]
```

min	21.0
10%	23.0
15%	23.0
33%	25.0
50%	27.0
59.2%	28.0
85%	31.0
90%	32.0
max	36.0

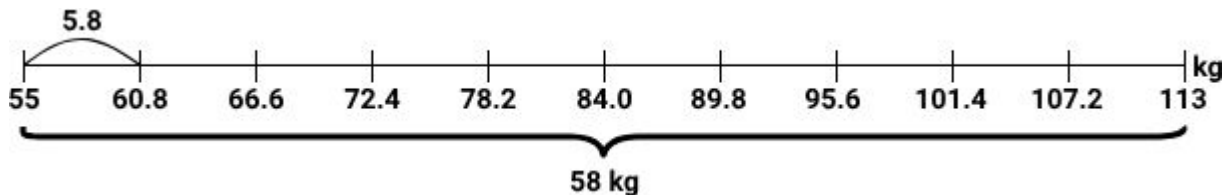
Name: Age, dtype: float64

0%	25%	50%	75%	100%
25% of ages are 24 or less 50% of ages are 27 or less 75% of ages are 30 or less 100% of ages are 36 or less				
21	24	27	30	36
↑ Minimum age in our data set				↑ Maximum age in our data set

Grouped Frequency Distribution Tables

```
>> wnba['Weight'].value_counts(bins = 10).sort_index()
```

(54.941, 60.8]	5
(60.8, 66.6]	21
(66.6, 72.4]	10
(72.4, 78.2]	33
(78.2, 84.0]	31
(84.0, 89.8]	24
(89.8, 95.6]	10
(95.6, 101.4]	3
(101.4, 107.2]	2
(107.2, 113.0]	3



Information Loss

```
>> wnba['PTS'].value_counts(bins = 10)
```

```
(1.417, 60.2]      30
```

```
(60.2, 118.4]     24
```

```
(118.4, 176.6]    17
```

```
(176.6, 234.8]    20
```

```
(234.8, 293.0]    17
```

```
(293.0, 351.2]     8
```

```
(351.2, 409.4]    10
```

```
(409.4, 467.6]     8
```

```
(467.6, 525.8]     4
```

```
(525.8, 584.0]     5
```

```
Name: PTS, dtype: int64
```

```
wnba['PTS'].value_counts(bins = 5).sort_index()
```

```
(1.417, 118.4]      54
```

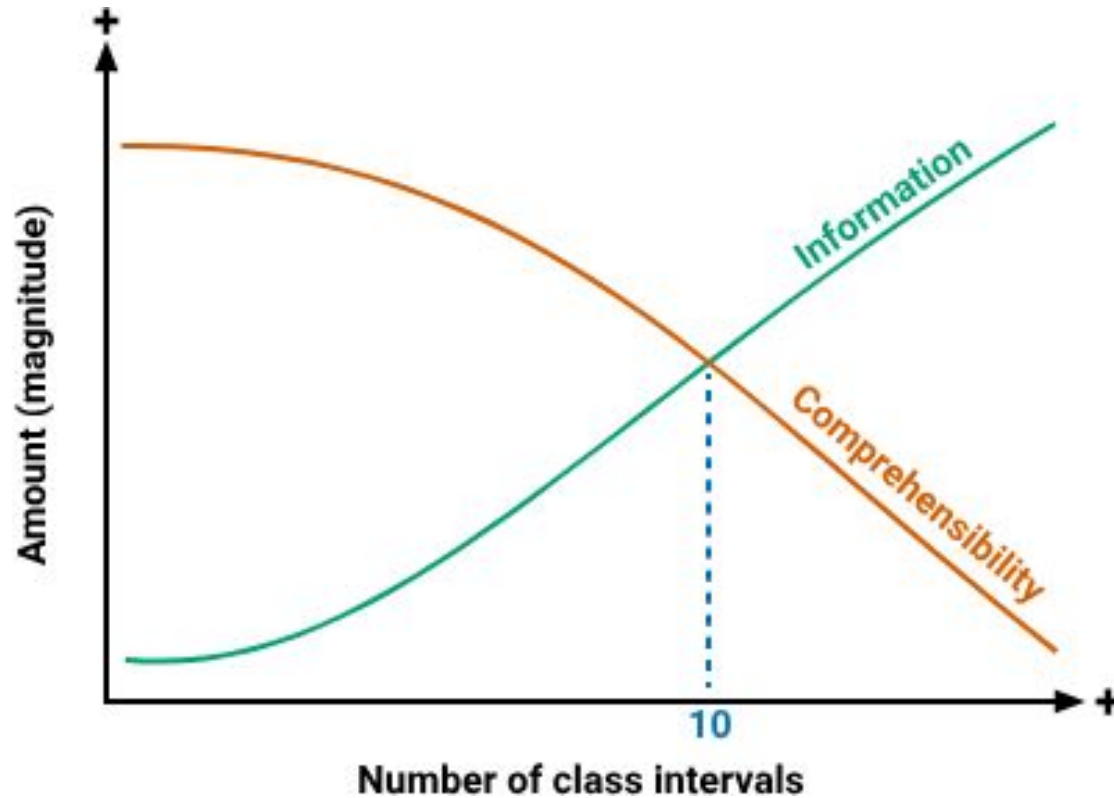
```
(118.4, 234.8]      37
```

```
(234.8, 351.2]      25
```

```
(351.2, 467.6]      18
```

```
(467.6, 584.0]       9
```

Information Loss



Readability for Grouped Frequency Tables

```
wnba[ 'PTS' ].value_counts(bins = 5).sort_index()
```

(1.417, 118.4]	54			
(118.4, 234.8]	37		(0, 100]	49
(234.8, 351.2]	25		(100, 200]	28
(351.2, 467.6]	18		(200, 300]	32
(467.6, 584.0]	9		(300, 400]	17
			(400, 500]	10
			(500, 600]	7

Name: PTS, dtype: int64

Readability for Grouped Frequency Tables

```
>> intervals = pd.interval_range(start = 0, end = 600, freq = 100)
>> intervals
IntervalIndex([(0, 100], (100, 200], (200, 300], (300, 400], (400, 500], (500, 600]]
              closed='right',
              dtype='interval[int64]')

>> gr_freq_table = pd.Series([0,0,0,0,0,0], index = intervals)
>> gr_freq_table
(0, 100]      0
(100, 200]    0
(200, 300]    0
(300, 400]    0
(400, 500]    0
(500, 600]    0
dtype: int64

>> for value in wnba['PTS']:
    for interval in intervals:
        if value in interval:
            gr_freq_table.loc[interval] += 1
            break
```

```
>> gr_freq_table
(0, 100]      49
(100, 200]    28
(200, 300]    32
(300, 400]    17
(400, 500]    10
(500, 600]     7
```

