

Formato para Trabajar en Colab 2025

Escribe el Nombre del Curso

Escribe tu nombre completo

Email:

Celular:

Código Estudiantil

✓ Temario

- ✓ 1. [Librerías para Trabajar en R y Python](#)
2. [Base de Datos - Estudiando la base de datos - DataMinig](#)
3. [Estadísticas Descriptivas \(Variables Numéricas\)](#)
4. [Variables Cuantitativas - Gráficas BoxPlot e Histograma](#)
5. [Verificando la Normalidad](#)
6. [Tomando una muestra aleatoria de tamaño 50](#)
7. [Tomando mil muestras - Bootstraps para una variable](#)
8. [Tomando mil muestras - Bootstraps para Varias variables](#)
9. [Datos Bivariados - DEFINITIVA vs SEXO](#)
10. [Datos Bivariados - DEFINITIVA vs ESTRATO](#)
11. [Datos Multivariados - DEFINITIVA vs ESTRATO vs SEXO](#)
12. [Intervalos de Confianza para la Media usando \$Z\$ y \$t\$ Student](#)
13. [Salvando la Base de Datos a Exportar nuestro Dataframes a limpios](#)
14. [Reclamando los datos](#)
15. [Regresión Lineal Simple](#)
16. [Matriz de correlación](#)
17. [Metodos de Regresión Lineal - `optimize.curve_fit`](#)
18. [Regresión lineal - MINIMOS CUADRADOS - Least Squares](#)
19. [Regresión lineal - Machine Learning](#)
20. [Regresión Lineal con Bootstrap y Diagrama de Dispersión](#)
21. [Un solo código para tres métodos de regresión lineal](#)
22. [Los cuatro métodos de regresión vistos, los parámetros calculados, AIC y el valor de \$R^2\$](#)
23. [Muestreo de Medias y Varianzas de una Variable](#)
24. [La simulación es para EDAD de distribución de medias y varianzas](#)
25. [Datos Categóricos - Diagrama de Barras](#)
26. [Datos agrupados para variables cuantitativa continuas](#)

27. [Histogramas y Polígonos de Frecuencia absolutas](#)
28. [Histogramas y Polígonos de Frecuencia Relativas](#)
29. [Histograma de frecuencias acumulado y Ojiva de los datos](#)
30. [Medidas de tendencia central agrupadas, Media, Mediana y Moda](#)
- 31.

[ULTIMO](#)

[1 Volver al inicio](#)

✓ 1. Librerías para Trabajar en R y Python

✓ Librerías para Trabajar en R

```
# @title **Librerías para Trabajar en R**
import rpy2.rinterface_lib.callbacks as rcb
import logging

# Desactivar warnings de R en Python
rcb.logger.setLevel(logging.ERROR) # Solo muestra errores, no warnings
%load_ext rpy2.ipython
```

✓ Librerías para Trabajar en Python

```
# @title **Librerías para Trabajar en Python**
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy import stats
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
from sympy.functions.combinatorial.factorials import factorial
import math
from math import sqrt
from numpy.ma.core import log
from sympy import integrate, init_printing
from sympy.abc import x
print("Setup Complete")
```

Setup Complete

[1 Volver al inicio](#)

✓ 2. Base de Datos - Estudiando la base de datos - DataMinig

✓ A1. Nuestra Base de datos en Python

```
# @title **A1. Nuestra Base de datos en Python**
url = 'https://raw.githubusercontent.com/JSEFERINO/Teoria-de-Probabilidad-MEYCD/main/DATOS202460ULTIMOS.csv'
df = pd.read_csv(url, delimiter=';')
df
```



	CURSO	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	NRC	PROGRAMA	EDAD	PESO	ESTATURA	...	A: Facilidad para aprender cosas nuevas	B: Memoria y atención	C: Relacionar tu experiencias con lo que aprendes	D: Autoestima	E: Apre
0	PROBABILIDAD	100	90	3.6	4.30	2314	F_NEGOCIOS	20	55	160	...	BAJO	ALTO	MEDIO	ALTO	
1	ESTADISTICA	70	75	0.9	2.50	1136	DERECHO	18	80	185	...	ALTO	ALTO	ALTO	ALTO	
2	PROBABILIDAD	85	95	3.9	3.80	2314	F_NEGOCIOS	19	60	158	...	BAJO	BAJO	BAJO	BAJO	
3	PROBABILIDAD	5	5	2.9	0.50	2314	MECANICA	18	72	181	...	MEDIO	ALTO	BAJO	MEDIO	
4	ESTADISTICA	20	70	3.7	0.55	1009	PSICOLOGÍA	19	45	163	...	ALTO	MEDIO	MEDIO	ALTO	
...
69	PROBABILIDAD	90	90	2.3	2.10	2314	F_NEGOCIOS	18	59	176	...	MEDIO	MEDIO	MEDIO	ALTO	
70	PROBABILIDAD	85	95	2.0	3.10	2314	F_NEGOCIOS	18	60	171	...	BAJO	BAJO	ALTO	ALTO	
71	ESTADISTICA	65	75	1.7	2.95	1136	DERECHO	20	55	164	...	BAJO	MEDIO	MEDIO	ALTO	
72	ESTADISTICA	100	100	2.3	3.20	2313	PSICOLOGÍA	19	67	171	...	BAJO	BAJO	BAJO	BAJO	
73	ESTADISTICA	100	100	3.8	3.10	1136	DERECHO	18	60	165	...	ALTO	ALTO	BAJO	ALTO	

74 rows × 26 columns

✓ A2. Nuestra Base de datos en R

```
# @title **A2. Nuestra Base de datos en R**
%%R
url <- 'https://raw.githubusercontent.com/JSEFERINO/Teoria-de-Probabilidad-MEYCD/main/DATOS202460ULTIMOS.csv'
df_R <- read.csv(url, sep = ';')

# Para visualizar las primeras filas del dataframe
head(df_R)
```



	CURSO	ASISTENCIA2	ASISTENCIA1	PARCIAL.1	PARCIAL.2	NRC	PROGRAMA	EDAD
1	PROBABILIDAD	100	90	3.6	4.30	2314	F_NEGOCIOS	20
2	ESTADISTICA	70	75	0.9	2.50	1136	DERECHO	18
3	PROBABILIDAD	85	95	3.9	3.80	2314	F_NEGOCIOS	19
4	PROBABILIDAD	5	5	2.9	0.50	2314	MECANICA	18
5	ESTADISTICA	20	70	3.7	0.55	1009	PSICOLOGÍA	19
6	ESTADISTICA	100	100	0.9	2.95	2313	PSICOLOGÍA	20
	PESO	ESTATURA	SEXO	ESTADO_CIVIL	ESTRATO		URBANO	
1	55	160	Femenino	SOLTERO (A)	II		Cartagena	
2	80	185	Masculino	SOLTERO (A)	III		Cartegena	
3	60	158	Femenino	SOLTERO (A)	III		Cartagena	
4	72	181	Masculino	SOLTERO (A)	V		Bolivar	
5	45	163	Femenino	SOLTERO (A)	II		Cartagena	
6	64	169	Femenino	SOLTERO (A)	I		Cartagena de Indias	
	TRANSPORTE		GR_SANGUINEO					
1	Transcaribe		0 positivo (0+)					
2	El bus que me deja mas cerca		A positivo (A +)					
3	Transcaribe		0 positivo (0+)					
4	Particular		0 positivo (0+)					
5	Mototaxi		0 positivo (0+)					
6	Transcaribe		0 positivo (0+)					
	A..Facilidad.para.aprender.cosas.nuevas		B..Memoria.y.atención					
1			BAJO				ALTO	
2			ALTO				ALTO	
3			BAJO				BAJO	
4			MEDIO				ALTO	
5			ALTO				MEDIO	
6			BAJO				BAJO	
	C..Relacionar.tu.experiencias.con.lo.que.aprendes		D..Autoestima					
1			MEDIO				ALTO	
2			ALTO				ALTO	
3			BAJO				BAJO	
4			BAJO				MEDIO	
5			MEDIO				ALTO	
6			BAJO				BAJO	
	E..Actitud.hacia.el.Aprendizaje		F..Ambiente.Familiar.para.estudiar					
1			ALTO				BAJO	
2			ALTO				ALTO	
3			MEDIO				MEDIO	
4			MEDIO				ALTO	
5			ALTO				BAJO	
6			MEDIO				MEDIO	
	G..Ansiedad.académica		H..Recursos.Educativos					
1			BAJO				BAJO	
2			MEDIO				MEDIO	
3			MEDIO				MEDIO	
4			MEDIO				ALTO	
5			MEDIO				BAJO	
6			MEDIO				MEDIO	
	I..Mentalidad.para.superar.adversidades		K..Regularidad.en.el.estudio					
1			ALTO				ALTO	
2			MEDIO				MEDIO	
3			MEDIO				MEDIO	
4			MEDIO				MEDIO	
5			ALTO				BAJO	

▼ B1. Cantidad de estudiantes y variables en Python

```
# @title **B1. Cantidad de estudiantes y variables en Python**
No_estudiantes_y_Variables = df.shape
print('(No estudiantes, No Variables) = ',No_estudiantes_y_Variables)
```

➞ (No estudiantes, No Variables) = (74, 26)

▼ B2. Cantidad de estudiantes y variables en R

```
# @title **B2. Cantidad de estudiantes y variables en R**
%%R
No_estudiantes_y_Variables <- dim(df_R)
cat('(No estudiantes, No Variables) = (', No_estudiantes_y_Variables[1], ', ', No_estudiantes_y_Variables[2], ')', sep = "")
```

➞ (No estudiantes, No Variables) = (74, 26)

▼ C1. Mostrar información del tipo de las variables y registros nulos (null) en Python

```
# @title **C1. Mostrar información del tipo de las variables y registros nulos (null) en Python**
df.info()
```

➞

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 74 entries, 0 to 73
Data columns (total 26 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   CURSO                                     74 non-null     object
1   ASISTENCIA2                             74 non-null     int64
2   ASISTENCIA1                             74 non-null     int64
3   PARCIAL 1                               74 non-null     float64
4   PARCIAL 2                               74 non-null     float64
5   NRC                                       74 non-null     int64
6   PROGRAMA                                 74 non-null     object
7   EDAD                                     74 non-null     int64
8   PESO                                    74 non-null     int64
9   ESTATURA                              74 non-null     int64
10  SEXO                                    74 non-null     object
11  ESTADO_CIVIL                           74 non-null     object
12  ESTRATO                                  73 non-null     object
13  URBANO                                  74 non-null     object
14  TRANSPORTE                             74 non-null     object
15  GR_SANGUINEO                           74 non-null     object
16  A: Facilidad para aprender cosas nuevas 74 non-null     object
17  B: Memoria y atención                   74 non-null     object
18  C: Relacionar tu experiencias con lo que aprendes 74 non-null     object
19  D: Autoestima                           74 non-null     object
20  E: Actitud hacia el Aprendizaje         74 non-null     object
21  F: Ambiente Familiar para estudiar      74 non-null     object
22  G: Ansiedad académica                   74 non-null     object
23  H: Recursos Educativos                   74 non-null     object
24  I: Mentalidad para superar adversidades 74 non-null     object
25  K: Regularidad en el estudio             74 non-null     object
dtypes: float64(2), int64(6), object(18)
memory usage: 15.2+ KB
```

▼ C2. Mostrar información del tipo de las variables y registros nulos (null) en R

```
# @title **C2. Mostrar información del tipo de las variables y registros nulos (null) en R**
%%R
# Opción 1: Usando str() para ver estructura y tipos de datos
cat("Información de la estructura del dataframe:\n")
str(df_R)

# Opción 2: Resumen más detallado con summary()
cat("\nResumen estadístico y conteo de NAs:\n")
summary(df_R)


# Opción 3: Conteo específico de valores nulos (NA en R)
cat("\nConteo de valores NA por variable:\n")
sapply(df_R, function(x) sum(is.na(x)))
```

➞

```
0
NRC
0
PROGRAMA
0
EDAD
0
PESO
0
ESTATURA
0
SEXO
0
ESTADO_CIVIL
0
ESTRATO
0
URBANO
0
TRANSPORTE
0
GR_SANGUINEO
0
A..Facilidad.para.aprender.cosas.nuevas
0
B..Memoria.y.atención
0
C..Relacionar.tu.experiencias.con.lo.que.aprendes
0
D..Autoestima
0
E..Actitud.hacia.el.Aprendizaje
0
F..Ambiente.Familiar.para.estudiar
0
G..Ansiedad.académica
0
H..Recursos.Educativos
0
I..Mentalidad.para.superar.adversidades
0
K..Regularidad.en.el.estudio
0
```


▼ D1. Mostrar los diferentes tipos de cursos en Python

```
# @title **D1. Mostrar los diferentes tipos de cursos en Python**
Tipos_cursos = df['CURSO'].unique()
print('Cursos=', Tipos_cursos)
```

 Cursos = ['PROBABILIDAD' 'ESTADISTICAI']

▼ D2. Mostrar los diferentes tipos de cursos en R


```
# @title **D2. Mostrar los diferentes tipos de cursos en R**
%%R
Tipos_cursos <- unique(df_R$CURSO)
print(paste('Cursos = ', paste(Tipos_cursos, collapse = ", ")))
```

 [1] "Cursos = PROBABILIDAD, ESTADISTICAI"

```
# Opción 1: Usando cat() para mejor formato
%%R
cat("Cursos:", paste(unique(df_R$CURSO), collapse = ", "), "\n")


# Opción 2: Con dplyr (tidyverse)
library(dplyr)
Tipos_cursos <- df_R %>% distinct(CURSO) %>% pull()
cat("Cursos:", Tipos_cursos, "\n")

# Opción 3: Mostrar como lista vertical
cat("Los cursos disponibles son:\n")
print(unique(df_R$CURSO))
```

 Cursos: PROBABILIDAD, ESTADISTICAI
Cursos: PROBABILIDAD ESTADISTICAI
Los cursos disponibles son:
[1] "PROBABILIDAD" "ESTADISTICAI"

▼ E1. Mostrar los diferentes tipos de programas en Python

```
# @title **E1. Mostrar los diferentes tipos de programas en Python**
Tipos_programas = df['PROGRAMA'].unique()
print('Programas=', Tipos_programas)
```

 Programas = ['F_NEGOCIOS' 'DERECHO' 'MECANICA' 'PSICOLOGÍA' 'C_DATOS' 'SISTEMAS'
'BIOMEDICA' 'INDUSTRIAL' 'ECONOMIA' 'MECATRONICA' 'NAVAL' 'C_SOCIAL'
'QUIMICA' 'ELECTRICA' 'CIVIL' 'CONTADURIA']

▼ E2. Mostrar los diferentes tipos de programas en R

```
# @title **E2. Mostrar los diferentes tipos de programas en R**
%%R
cat("Programas = ", paste(unique(df_R$PROGRAMA), collapse = ", "), "\n")
```

Programas = F_NEGOCIOS, DERECHO, MECANICA, PSICOLOGÍA, C_DATOS, SISTEMAS, BIOMEDICA, INDUSTRIAL, ECONOMIA, MECATRONICA, NAVAL, C_SOCIAL, QUIMICA, ELECTRICA, C

▼ F1. Mostrar los diferentes tipos de estratos en Python

```
# @title **F1. Mostrar los diferentes tipos de estratos en Python**
Tipos_estratos = df['ESTRATO'].unique()
print('Estratos=', Tipos_estratos)
```

Estratos= ['II' 'III' 'V' 'I' 'IV' nan]

▼ G1. Mostrar la posicion de los estudiantes con "nan" en estrato en Python

```
# @title **G1. Mostrar la posicion de los estudiantes con "nan" en estrato en Python**
nan_estrato_positions = df[df['ESTRATO'].isnull()].index.tolist()
print("Posiciones de estudiantes con 'nan' en estrato:", nan_estrato_positions)
```

Posiciones de estudiantes con 'nan' en estrato: [17]

▼ G2. Mostrar la posición de los estudiantes con NA en estrato en R

```
# @title **G2. Mostrar la posición de los estudiantes con NA en estrato en R**
%%R
nan_estrato_positions <- which(is.na(df_R$ESTRATO))
print(paste("Posiciones de estudiantes con NA en estrato:",
  paste(nan_estrato_positions, collapse = ", ")))
```

[1] "Posiciones de estudiantes con NA en estrato: "

▼ F. Mostrar la posicion de los estudiantes con "nan" en estrato

```
# @title **F. Mostrar la posicion de los estudiantes con "nan" en estrato**
df['ESTRATO'].head(20)
```

	ESTRATO
0	II
1	III
2	III
3	V
4	II
5	I
6	IV
7	III
8	III
9	I
10	I
11	II
12	II
13	I
14	IV
15	IV
16	III
17	NaN
18	I
19	V

dtype: object

%%R

df_R\$ESTRATO

```

[1] "II"  "III" "III" "V"  "II"  "I"   "IV"  "III" "III" "I"   "I"   "II"
[13] "II"  "I"   "IV"  "IV"  "III" ""    "I"   "V"   "II"  "I"   "II"  "II"
[25] "III" "II"  "III" "II"  "IV"  "I"   "III" "IV"  "I"   "III" "II"  "II"
[37] "III" "III" "I"   "IV"  "I"   "II"  "V"   "II"  "IV"  "I"   "III" "II"
[49] "III" "V"   "III" "II"  "III" "II"  "III" "II"  "III" "II"  "II"  "II"
[61] "II"  "IV"  "I"   "V"   "II"  "IV"  "II"  "II"  "I"   "IV"  "II"  "III"
[73] "I"   "I"

```

▼ G. Mostrar la posicion de los estudiantes con "nan" en estrato

```

# @title **G. Mostrar la posicion de los estudiantes con "nan" en estrato**

df['ESTRATO'].fillna('II', inplace=True)
print(df['ESTRATO'].unique())

[ 'II' 'III' 'V' 'I' 'IV' ]

```

▼ H. Mostrar los diferentes tipos de transporte

```

# @title **H. Mostrar los diferentes tipos de transporte**
Tipos_transporte = df['TRANSPORTE'].unique()
print('Transporte = ', Tipos_transporte)

Transporte = [ 'Transcribe' 'El bus que me deja mas cerca' 'Particular' 'Mototaxi'
               'Taxi' ]

```

▼ I. Cambia en TRANSPORTE "El bus que me deja mas cerca" por "Bus"

```

# @title **I. Cambia en TRANSPORTE "El bus que me deja mas cerca" por "Bus"

df['TRANSPORTE'] = df['TRANSPORTE'].replace('El bus que me deja mas cerca', 'Bus')
print('Transporte = ', df['TRANSPORTE'].unique())

Transporte = [ 'Transcribe' 'Bus' 'Particular' 'Mototaxi' 'Taxi' ]

```

▼ J. Mostrar los diferentes tipos de grupos sanguíneos

```

# @title **J. Mostrar los diferentes tipos de grupos sanguíneos**
Tipos_sangre = df['GR_SANGUINEO'].unique()
print('Grupo Sanguíneo=', Tipos_sangre)

Grupo Sanguíneo= [ 'O positivo (O+)' 'A positivo (A +)' 'B positivo (B +)' 'O negativo (O-)' ]

```

▼ K. Cambia en TRANSPORTE "El bus que me deja mas cerca" por "Bus"

```

# @title **K. Cambia en TRANSPORTE "El bus que me deja mas cerca" por "Bus"

df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('O positivo (O+)', 'O_POS')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('A positivo (A +)', 'A_POS')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('B positivo (B +)', 'B_POS')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('AB positivo (AB +)', 'AB_POS')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('O negativo (O -)', 'O_NEG')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('A negativo (A -)', 'A_NEG')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('B negativo (B -)', 'B_NEG')
df['GR_SANGUINEO'] = df['GR_SANGUINEO'].replace('AB negativo (AB -)', 'AB_NEG')
print('Grupo Sanguíneo=', df['GR_SANGUINEO'].unique())

Grupo Sanguíneo= [ 'O_POS' 'A_POS' 'B_POS' 'O negativo (O-)' ]

```

▼ L. Verificar la cantidad de datos faltantes en cada columna

```

# @title **L. Verificar la cantidad de datos faltantes en cada columna**
faltantes = df.isnull().sum()
faltantes

```



	0
CURSO	0
ASISTENCIA2	0
ASISTENCIA1	0
PARCIAL 1	0
PARCIAL 2	0
NRC	0
PROGRAMA	0
EDAD	0
PESO	0
ESTATURA	0
SEXO	0
ESTADO_CIVIL	0
ESTRATO	0
URBANO	0
TRANSPORTE	0
GR_SANGUINEO	0
A: Facilidad para aprender cosas nuevas	0
B: Memoria y atención	0
C: Relacionar tu experiencias con lo que aprendes	0
D: Autoestima	0
E: Actitud hacia el Aprendizaje	0
F: Ambiente Familiar para estudiar	0
G: Ansiedad académica	0
H: Recursos Educativos	0
I: Mentalidad para superar adversidades	0
K: Regularidad en el estudio	0

dtype: int64

[Volver al inicio](#)

3. Estadísticas Descriptivas (Variables Numéricas)

A. Estadísticas descriptivas generales

```
# @title **A. Estadísticas descriptivas generales**
stats_descriptivas = df.describe()
# Mostrar estadísticas descriptivas
stats_descriptivas
```



	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	NRC	EDAD	PESO	ESTATURA
count	74.000000	74.000000	74.000000	74.000000	74.000000	74.000000	74.000000	74.000000
mean	88.243243	86.756757	2.844595	3.114865	1549.851351	18.702703	63.324324	168.391892
std	17.564605	15.112193	0.994112	0.871022	619.364161	1.190484	11.354931	8.533283
min	5.000000	5.000000	0.900000	0.500000	1009.000000	17.000000	42.000000	153.000000
25%	85.000000	80.000000	2.000000	2.900000	1010.000000	18.000000	55.000000	163.000000
50%	90.000000	90.000000	2.850000	3.150000	1136.000000	18.000000	62.000000	168.000000
75%	100.000000	95.000000	3.600000	3.687500	2314.000000	19.000000	70.000000	174.000000
max	100.000000	100.000000	4.900000	4.650000	2314.000000	22.000000	102.000000	192.000000

B. Estadísticas descriptivas para variables categóricas

```
# @title **B. Estadísticas descriptivas para variables categóricas**
categ_stats = df[['CURSO', 'PROGRAMA', 'URBANO', 'TRANSPORTE', 'SEXO', 'ESTRATO', 'GR_SANGUINEO']].describe()

# Mostrar estadísticas descriptivas
categ_stats
```




	CURSO	PROGRAMA	URBANO	TRANSPORTE	SEXO	ESTRATO	GR_SANGUINEO
count	74	74	74	74	74	74	74
unique	2	16	28	5	2	5	4
top	PROBABILIDAD	F_NEGOCIOS	Cartagena	Transcribe	Femenino	II	O_POS
freq	48	16	17	35	42	26	44

▼ C. Estadísticas descriptivas para variables categóricas - URBANO

```
# @title **C. Estadísticas descriptivas para variables categóricas - URBANO**
df['URBANO'].unique()
```



```
array(['Cartagena ', 'Cartegena', 'Cartagena', 'Bolívar',
      'Cartagena de Indias ', 'Bolívar ', 'pontezuella', 'CARTAGENA',
      'cartagena', 'Turbaco', 'Parque Heredia', 'Zaragocilla ',
      'cartagena de indias ', bolivar', 'Cartagena de indias ',
      'El Carmen de Bolívar ', 'Barranco de loba', 'Pontezuella ',
      'Turbaco/Bolívar', 'Cartagena (barrio: pie de la popa)',
      'Boquilla ', 'San José de los campanos', 'bolivar', 'Turbaco ',
      'Bolíbar Cartagena', 'Arjona ', 'Bolívar ', 'El rodeo',
      'Villa de la cruz'], dtype=object)
```

▼ D. Redefiniendo una variable - Una nueva Variable

```
# @title **D. Redefiniendo una variable - Una nueva Variable**
df['DEFINITIVA'] = df['PARCIAL 1']
df
```



	CURSO	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	NRC	PROGRAMA	EDAD	PESO	ESTATURA	...	B: Memoria y atención	C: Relacionar tu experiencias con lo que aprendes	D: Autoestima	E: Actitud hacia el Aprendizaje	An Fz
0	PROBABILIDAD	100	90	3.6	4.30	2314	F_NEGOCIOS	20	55	160	...	ALTO	MEDIO	ALTO	ALTO	
1	ESTADISTICA	70	75	0.9	2.50	1136	DERECHO	18	80	185	...	ALTO	ALTO	ALTO	ALTO	
2	PROBABILIDAD	85	95	3.9	3.80	2314	F_NEGOCIOS	19	60	158	...	BAJO	BAJO	BAJO	MEDIO	
3	PROBABILIDAD	5	5	2.9	0.50	2314	MECANICA	18	72	181	...	ALTO	BAJO	MEDIO	MEDIO	
4	ESTADISTICA	20	70	3.7	0.55	1009	PSICOLOGÍA	19	45	163	...	MEDIO	MEDIO	ALTO	ALTO	
...
69	PROBABILIDAD	90	90	2.3	2.10	2314	F_NEGOCIOS	18	59	176	...	MEDIO	MEDIO	ALTO	ALTO	
70	PROBABILIDAD	85	95	2.0	3.10	2314	F_NEGOCIOS	18	60	171	...	BAJO	ALTO	ALTO	ALTO	
71	ESTADISTICA	65	75	1.7	2.95	1136	DERECHO	20	55	164	...	MEDIO	MEDIO	ALTO	ALTO	
72	ESTADISTICA	100	100	2.3	3.20	2313	PSICOLOGÍA	19	67	171	...	BAJO	BAJO	BAJO	MEDIO	
73	ESTADISTICA	100	100	3.8	3.10	1136	DERECHO	18	60	165	...	ALTO	BAJO	ALTO	ALTO	

74 rows × 27 columns

[T Volver al inicio](#)

▼ 4. Variables Cuantitativas - Gráficas BoxPlot e Histograma

▼ A. Gráficas BoxPlot e Histograma: DEFINITIVA, EDAD, ESTATURA

```
# @title **A. Gráficas BoxPlot e Histograma: DEFINITIVA, EDAD, ESTATURA**
plt.figure(figsize=(8, 4))

# Histograma definitivas
plt.subplot(2, 3, 1)
plt.hist(df['DEFINITIVA'], bins=10, color='skyblue')
plt.title('Distribución de las Definitivas')
plt.xlabel('Definitiva (0->5)')
plt.ylabel('Frecuencia')

# Histograma de edades
plt.subplot(2, 3, 2)
plt.hist(df['EDAD'], bins=10, color='lightgreen')
plt.title('Distribución de Edad')
plt.xlabel('Edad (Años)')
plt.ylabel('Frecuencia')

# Boxplot de las definitivas
```

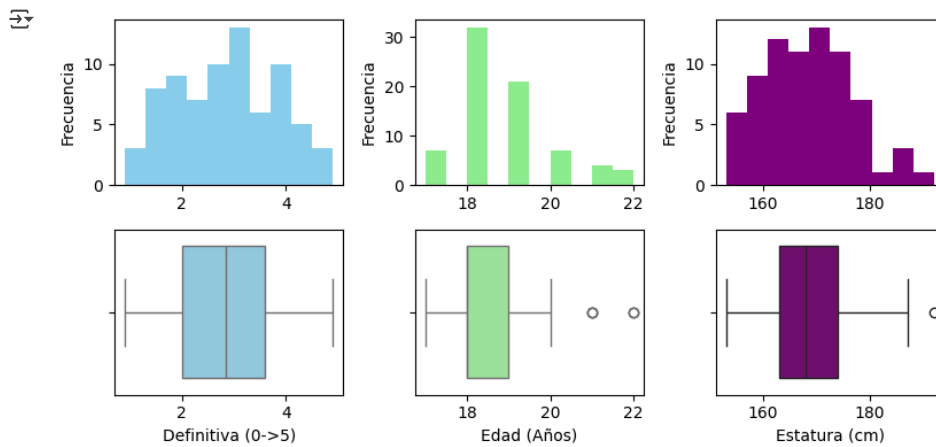
```
plt.subplot(2, 3, 4)
sns.boxplot(data=df, x='DEFINITIVA', color='skyblue')
plt.title('Distribución de Notas Definitivas')
plt.xlabel('Definitiva (0->5)')

# Boxplot Edad
plt.subplot(2, 3, 5)
sns.boxplot(data=df, x='EDAD', color='lightgreen')
plt.xlabel('Edad (Años)')
plt.title('Distribución de Notas Definitivas')

# Histograma de la estatura
plt.subplot(2, 3, 3)
plt.hist(df['ESTATURA'], bins=10, color='purple')
plt.title('Distribución de Asistencia')
plt.ylabel('Frecuencia')
plt.xlabel('Estatura (cm)')

# Boxplot de estatura
plt.subplot(2, 3, 6)
sns.boxplot(data=df, x='ESTATURA', color='purple')
plt.title('Distribución de la Asistencia')
plt.xlabel('Estatura (cm)')

#ajustar espaciado entre subplot
plt.tight_layout()
plt.show()
```



✓ B. Gráficas BoxPlot e Histograma:PARCIAL 2, PESO, ASISTENCIA1

```
# @title **B. Gráficas BoxPlot e Histograma:PARCIAL 2, PESO, ASISTENCIA1**
plt.figure(figsize=(8, 4))

# Histograma PARCIAL 2
plt.subplot(2, 3, 1)
plt.hist(df['PARCIAL 2'], bins=10, color='skyblue')
plt.title('Distribución de las Definitivas')
plt.xlabel('Definitiva (0->5)')
plt.ylabel('Frecuencia')

# Histograma de edades
plt.subplot(2, 3, 2)
plt.hist(df['PESO'], bins=10, color='lightgreen')
plt.title('Distribución de Edad')
plt.xlabel('Edad (Años)')
plt.ylabel('Frecuencia')

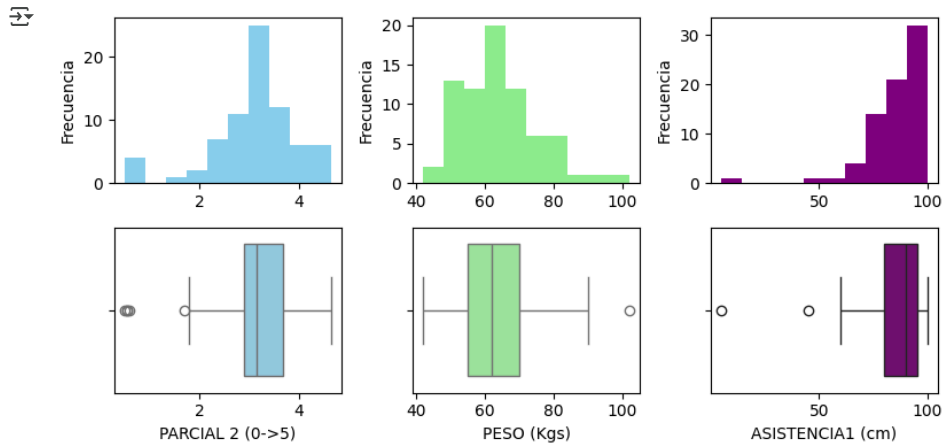
# Boxplot de las PARCIAL 2
plt.subplot(2, 3, 4)
sns.boxplot(data=df, x='PARCIAL 2', color='skyblue')
plt.title('Distribución de Notas Definitivas')
plt.xlabel('PARCIAL 2 (0->5)')

# Boxplot PESO
plt.subplot(2, 3, 5)
sns.boxplot(data=df, x='PESO', color='lightgreen')
plt.xlabel('PESO (Kgs)')
plt.title('Distribución de Notas Definitivas')

# Histograma de la ASISTENCIA1
plt.subplot(2, 3, 3)
plt.hist(df['ASISTENCIA1'], bins=10, color='purple')
plt.title('Distribución de Asistencia')
plt.ylabel('Frecuencia')
plt.xlabel('Estatura (cm)')
```

```
# Boxplot de ASISTENCIA1
plt.subplot(2, 3, 6)
sns.boxplot(data=df, x='ASISTENCIA1', color='purple')
plt.title('Distribución de la Asistencia')
plt.xlabel('ASISTENCIA1 (cm)')

#ajustar espaciado entre subplot
plt.tight_layout()
plt.show()
```



[Volver al inicio](#)

5. Verificando la Normalidad

A. Recordando nuestra variables

```
# @title **A. Recordando nuestra variables**
df.columns
```

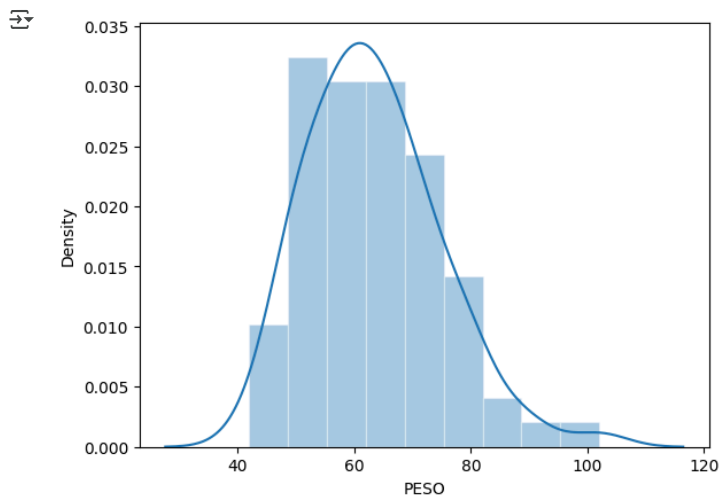
```
Index(['CURSO', 'ASISTENCIA2', 'ASISTENCIA1', 'PARCIAL 1', 'PARCIAL 2', 'NRC',
      'PROGRAMA', 'EDAD', 'PESO', 'ESTATURA', 'SEXO', 'ESTADO_CIVIL',
      'ESTRATO', 'URBANO', 'TRANSPORTE', 'GR_SANGUINEO',
      'A: Facilidad para aprender cosas nuevas', 'B: Memoria y atención',
      'C: Relacionar tu experiencias con lo que aprendes', 'D: Autoestima',
      'E: Actitud hacia el Aprendizaje', 'F: Ambiente Familiar para estudiar',
      'G: Ansiedad académica', 'H: Recursos Educativos',
      'I: Mentalidad para superar adversidades',
      'K: Regularidad en el estudio', 'DEFINITIVA'],
      dtype='object')
```

B. Histograma vs Densidad Normal - PESO

```
# @title **B. Histograma vs Densidad Normal - PESO**

sns.histplot(df['PESO'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)

plt.show()
```



✓ C. Histograma vs Densidad Normal - Todas las variables

```
# @title **C. Histograma vs Densidad Normal - Todas las variables**
plt.figure(figsize=(8, 4))

# Boxplot de las ASISTENCIA2
plt.subplot(2, 3, 1)
sns.histplot(df['ASISTENCIA2'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
plt.title('Distribución de Notas Definitivas')
plt.xlabel('ASISTENCIA2')

# Boxplot de las PARCIAL 1
plt.subplot(2, 3, 2)
sns.histplot(df['PARCIAL 1'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
plt.title('Distribución de Notas Definitivas')
plt.xlabel('PARCIAL 1')

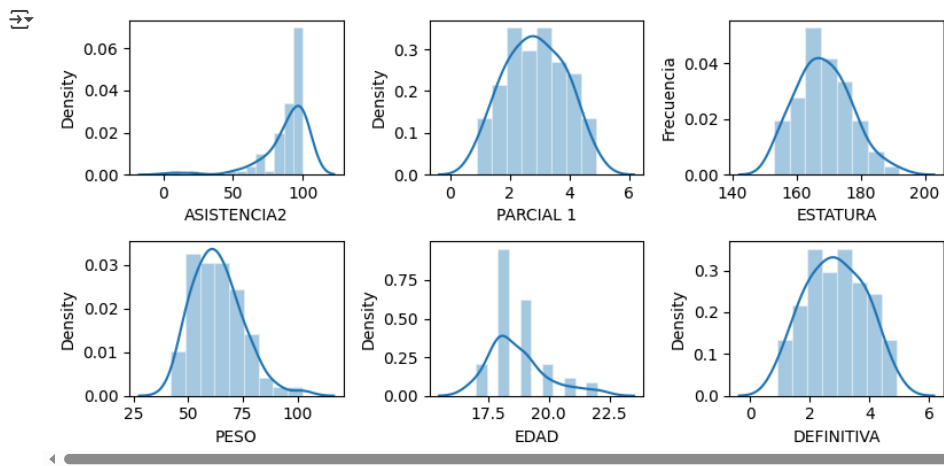
# Boxplot de las PESO
plt.subplot(2, 3, 4)
sns.histplot(df['PESO'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
plt.title('Distribución de Notas Definitivas')
plt.xlabel('PESO')

# Boxplot PESO
plt.subplot(2, 3, 5)
sns.histplot(df['EDAD'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
plt.title('Distribución de Notas Definitivas')

# Histograma de la ASISTENCIA1
plt.subplot(2, 3, 3)
sns.histplot(df['ESTATURA'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
plt.title('Distribución de Asistencia')
plt.ylabel('Frecuencia')
plt.xlabel('Estatura (cm)')

# Boxplot de ASISTENCIA1
plt.subplot(2, 3, 6)
sns.histplot(df['DEFINITIVA'], kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
plt.title('Distribución de la Asistencia')
plt.xlabel('DEFINITIVA')

#ajustar espaciado entre subplot
plt.tight_layout()
plt.show()
```



[Volver al inicio](#)

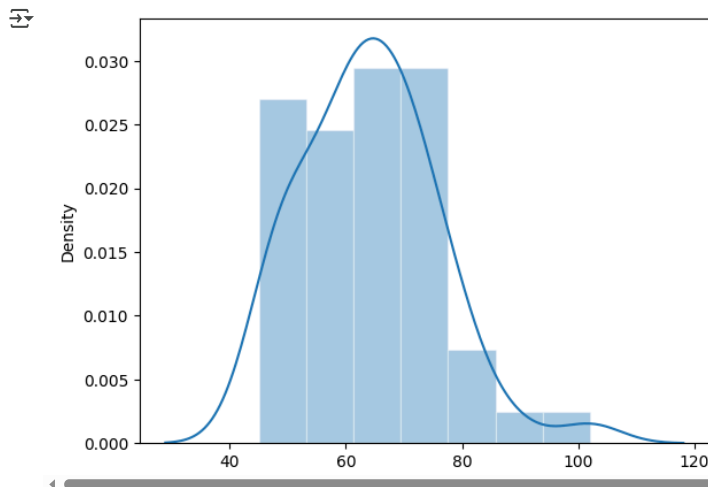
6. Tomando una muestra aleatoria

A. Tomando una muestra aleatoria

```
# @title **A. Tomando una muestra aleatoria**
tamano_muestra = 50
muestra_principal = np.random.choice(df['PESO'],
                                     tamano_muestra) # choice nos permmite sacar unos datos

sns.histplot(muestra_principal, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)

plt.show()
```



C. Histograma vs Densidad Normal - Todas las variables

```
# @title **C. Histograma vs Densidad Normal - Todas las variables**
plt.figure(figsize=(8, 4))

tamano_muestra = 50

# Boxplot de las ASISTENCIA2
plt.subplot(2, 3, 1)
muestra_principal1 = np.random.choice(df['ASISTENCIA2'],
                                     tamano_muestra) # choice nos permmite sacar unos datos

sns.histplot(muestra_principal1, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)

#plt.title('Distribución de Notas Definitivas')
plt.xlabel('ASISTENCIA2')

# Boxplot de las PARCIAL 1
plt.subplot(2, 3, 2)
muestra_principal2 = np.random.choice(df['PARCIAL 1'],
```

```

tamaño_muestra) # choice nos permmite sacar unos datos

sns.histplot(muestra_principal2, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, .4),)
#plt.title('Distribución de Notas Definitivas')
plt.xlabel('PARCIAL 1')

# Boxplot de las PESO
plt.subplot(2, 3, 4)
muestra_principal3 = np.random.choice(df['PESO'],
                                     tamaño_muestra) # choice nos permmite sacar unos datos

sns.histplot(muestra_principal3, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, .4),)
#plt.title('Distribución de Notas Definitivas')
plt.xlabel('PESO')

# Boxplot PESO
plt.subplot(2, 3, 5)
muestra_principal4 = np.random.choice(df['EDAD'],
                                     tamaño_muestra) # choice nos permmite sacar unos datos

sns.histplot(muestra_principal4, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, .4),)
#plt.title('Distribución de Notas Definitivas')
plt.xlabel('EDAD')

# Histograma de la ASISTENCIA1
plt.subplot(2, 3, 3)

muestra_principal5 = np.random.choice(df['ESTATURA'],
                                     tamaño_muestra) # choice nos permmite sacar unos datos

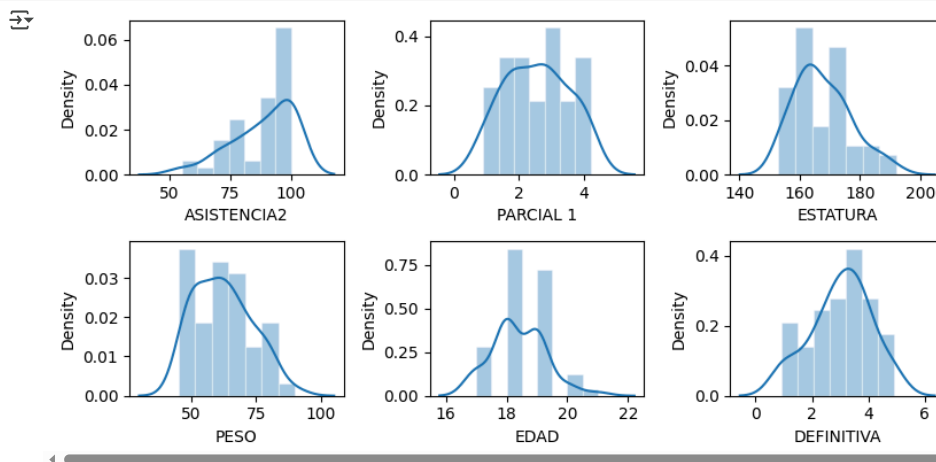
sns.histplot(muestra_principal5, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, .4),)
#plt.title('Distribución de Notas Definitivas')
plt.xlabel('ESTATURA')

# Boxplot de ASISTENCIA1
plt.subplot(2, 3, 6)
muestra_principal6 = np.random.choice(df['DEFINITIVA'],
                                     tamaño_muestra) # choice nos permmite sacar unos datos

sns.histplot(muestra_principal6, kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, .4),)
#plt.title('Distribución de Notas Definitivas')
plt.xlabel('DEFINITIVA')

#ajustar espaciado entre subplot
plt.tight_layout()
plt.show()

```



[Volver al inicio](#)

7. Tomando mil muestras - para una variable

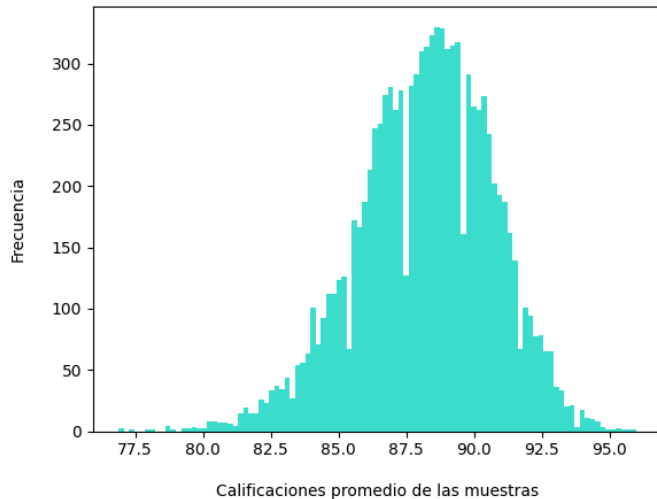
```

# **A. Usando la tecnica de BootStrap seleccionaremos 1000 muestras aleatorias y las guardaremos**
muestras1 = np.array([]) #En este espacio guardaremos cada muestra tomada
num_muestras = 10000 # Este será el tamaño de las muestra

```

```
for m in range(num_muestras): # Con este procedimiento garantizamos tomar las mil muestras
    muestras1 = np.append(muestras1,
                          np.random.choice(df['ASISTENCIA2'],
                                             tamaño_muestra,
                                             replace=True)) # guardamos en muestras las seleccionadas de la poblacion inicial
muestras1 = muestras1.reshape(-1, tamaño_muestra)
muestras1.shape, muestras1 # nos muestras el tamaño del archivo y las muestras

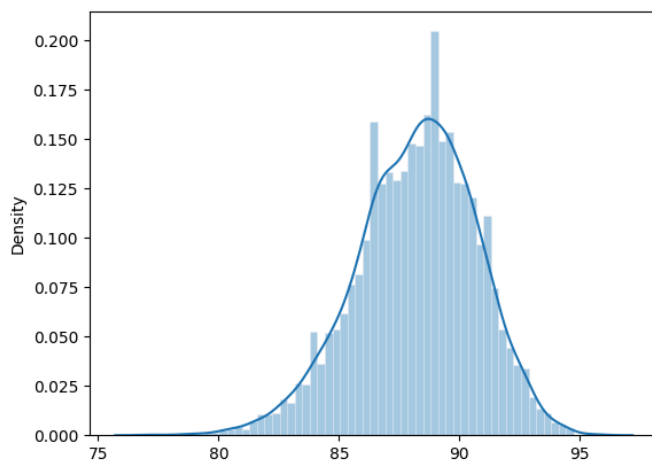
# **d. Estamos realizando el muestreo de medias**
plt.hist(muestras1.mean(axis=1), bins=100, color="turquoise") # Histograma de la distribución muestral de la media
plt.xlabel("\nCalificaciones promedio de las muestras")
plt.ylabel("Frecuencia\n")
plt.show()
```



```
sns.histplot(muestras1.mean(axis=1), kde=True, stat="density", kde_kws=dict(cut=3),
             alpha=.4, edgecolor=(1, 1, 1, .4),)
```



<Axes: ylabel='Density'>



[Volver al inicio](#)

8. Tomando mil muestras - Bootstraps para varias variables

A. Usando la técnica de Bootstrap seleccionaremos 1000 muestras aleatorias y las guardaremos

```
# @title **A. Usando la técnica de Bootstrap seleccionaremos 1000 muestras aleatorias y las guardaremos**
plt.figure(figsize=(12, 8))

# **A. Usando la técnica de Bootstrap seleccionaremos 1000 muestras aleatorias y las guardaremos**
# Inicializamos arrays para almacenar las medias de cada muestra
medias_asistencia = np.array([])
medias_parcial1 = np.array([])
medias_peso = np.array([])
medias_edad = np.array([])
medias_estatura = np.array([])
medias_definitiva = np.array([])

num_muestras = 1000 # Número de muestras bootstrap
```

```

tamano_muestra = len(df) # Tamaño de cada muestra (mismo que el dataset original)

for _ in range(num_muestras):
    # Muestra bootstrap para cada variable y calculamos su media
    muestra_asistencia = np.random.choice(df['ASISTENCIA2'], tamano_muestra, replace=True)
    medias_asistencia = np.append(medias_asistencia, np.mean(muestra_asistencia))

    muestra_parcial1 = np.random.choice(df['PARCIAL 1'], tamano_muestra, replace=True)
    medias_parcial1 = np.append(medias_parcial1, np.mean(muestra_parcial1))

    muestra_peso = np.random.choice(df['PESO'], tamano_muestra, replace=True)
    medias_peso = np.append(medias_peso, np.mean(muestra_peso))

    muestra_edad = np.random.choice(df['EDAD'], tamano_muestra, replace=True)
    medias_edad = np.append(medias_edad, np.mean(muestra_edad))

    muestra_estatura = np.random.choice(df['ESTATURA'], tamano_muestra, replace=True)
    medias_estatura = np.append(medias_estatura, np.mean(muestra_estatura))

    muestra_definitiva = np.random.choice(df['DEFINITIVA'], tamano_muestra, replace=True)
    medias_definitiva = np.append(medias_definitiva, np.mean(muestra_definitiva))

# Visualización de las distribuciones de las medias
# Histograma de ASISTENCIA2
plt.subplot(2, 3, 1)
sns.histplot(medias_asistencia, kde=True, stat="density",
              alpha=.4, edgecolor=(1, 1, 1, .4))
plt.xlabel('Media de ASISTENCIA2')
plt.ylabel('Densidad')

# Histograma de PARCIAL 1
plt.subplot(2, 3, 2)
sns.histplot(medias_parcial1, kde=True, stat="density",
              alpha=.4, edgecolor=(1, 1, 1, .4))
plt.xlabel('Media de PARCIAL 1')
plt.ylabel('Densidad')

# Histograma de PESO
plt.subplot(2, 3, 3)
sns.histplot(medias_peso, kde=True, stat="density",
              alpha=.4, edgecolor=(1, 1, 1, .4))
plt.xlabel('Media de PESO')
plt.ylabel('Densidad')

# Histograma de EDAD
plt.subplot(2, 3, 4)
sns.histplot(medias_edad, kde=True, stat="density",
              alpha=.4, edgecolor=(1, 1, 1, .4))
plt.xlabel('Media de EDAD')
plt.ylabel('Densidad')

# Histograma de ESTATURA
plt.subplot(2, 3, 5)
sns.histplot(medias_estatura, kde=True, stat="density",
              alpha=.4, edgecolor=(1, 1, 1, .4))
plt.xlabel('Media de ESTATURA')
plt.ylabel('Densidad')

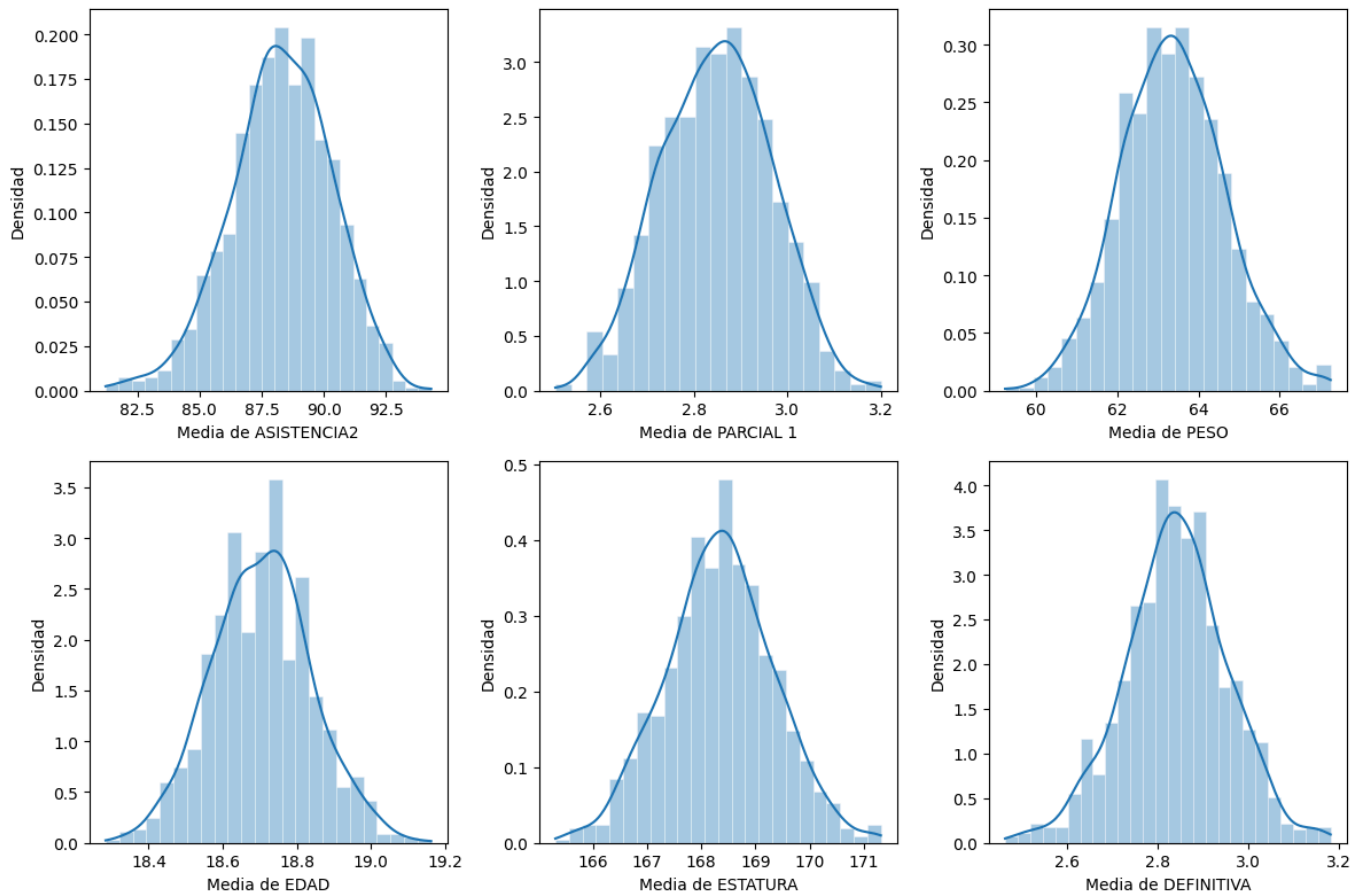
# Histograma de DEFINITIVA
plt.subplot(2, 3, 6)
sns.histplot(medias_definitiva, kde=True, stat="density",
              alpha=.4, edgecolor=(1, 1, 1, .4))
plt.xlabel('Media de DEFINITIVA')
plt.ylabel('Densidad')

# Ajustar espaciado entre subplots
plt.tight_layout()
plt.suptitle('Distribución de las medias mediante Bootstrap', y=1.02)
plt.show()

```




Distribución de las medias mediante Bootstrap

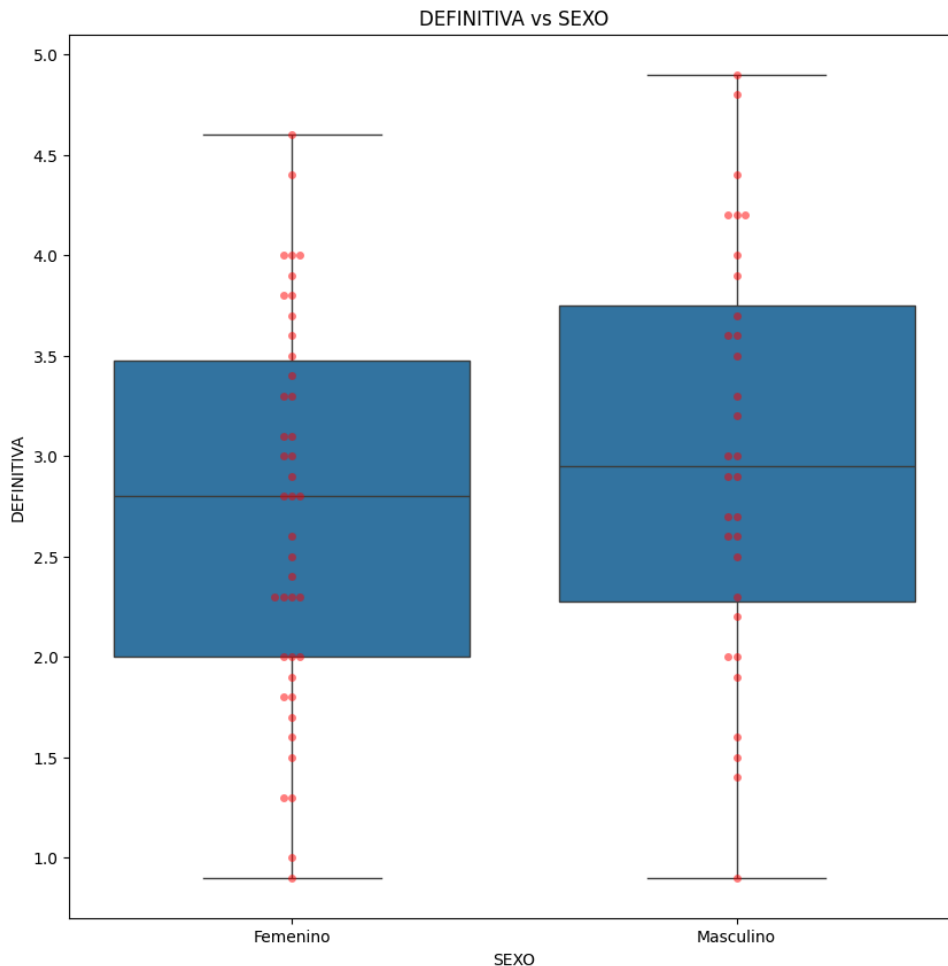

[Volver al inicio](#)

9. Datos Bivariados - DEFINITIVA vs SEXO

A. BoxPlot para DEFINITIVA vs SEXO

```
# @title **A. BoxPlot para DEFINITIVA vs SEXO**
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
ax.set_title('DEFINITIVA vs SEXO')
sns.boxplot(x="SEXO", y=df['DEFINITIVA'], data=df, ax=ax)
sns.swarmplot(x="SEXO", y="DEFINITIVA", data=df, color='red', alpha=0.5, ax=ax)
```

<Axes: title={'center': 'DEFINITIVA vs SEXO'}, xlabel='SEXO', ylabel='DEFINITIVA'>



▼ B. Descriptores numéricos Bivariados 'DEFINITIVA vs SEXO'

```
# @title **B. Descriptores numéricos Bivariados 'DEFINITIVA vs SEXO'**
print('DEFINITIVA vs SEXO')
df.groupby('SEXO')['DEFINITIVA'].agg(['mean', 'std', 'median', 'count'])
```

DEFINITIVA vs SEXO

	mean	std	median	count
SEXO				
Femenino	2.721429	0.960827	2.80	42
Masculino	3.006250	1.028917	2.95	32

▼ C. Coeficiente de variación - Descriptores numéricos Bivariados 'PROMEDIO vs SEXO'

```
# @title **C. Coeficiente de variación - Descriptores numéricos Bivariados 'PROMEDIO vs SEXO'**
print('DEFINITIVA vs SEXO')
df3 = df.groupby('SEXO')['DEFINITIVA'].agg(['mean', 'std', 'median', 'count'])
df3['coef_var(%)'] = round((df3['std'] / df3['mean']) * 100,1)
df3
```

DEFINITIVA vs SEXO

	mean	std	median	count	coef_var(%)
SEXO					
Femenino	2.721429	0.960827	2.80	42	35.3
Masculino	3.006250	1.028917	2.95	32	34.2

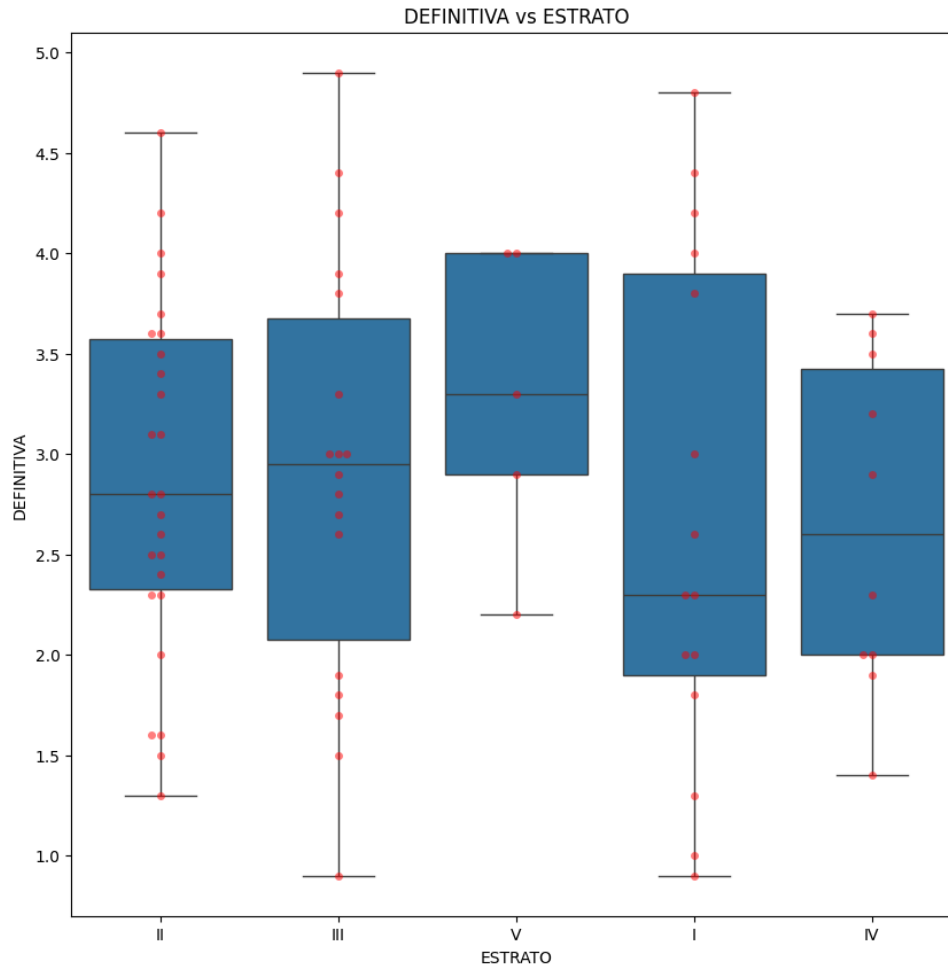
[1 Volver al inicio](#)

▼ 10. Datos Bivariados - DEFINITIVA vs ESTRATO

▼ A. Diagramaa de Caja Bivariado 'DEFINITIVA vs ESTRATO'

```
# @title **A. Diagramaa de Caja Bivariado 'DEFINITIVA vs ESTRATO'**
fig, ax = plt.subplots(1, 1, figsize=(10, 10))
ax.set_title('DEFINITIVA vs ESTRATO')
sns.boxplot(x="ESTRATO", y=df['DEFINITIVA'], data=df, ax=ax)
sns.swarmplot(x="ESTRATO", y="DEFINITIVA", data=df, color='red', alpha=0.5, ax=ax)
```

<Axes: title={'center': 'DEFINITIVA vs ESTRATO'}, xlabel='ESTRATO', ylabel='DEFINITIVA'>



▼ B. Descriptores numericos e intervalos de confianza 'DEFINITIVA vs ESTRATO'

```
# @title **B. Descriptores numericos e intervalos de confianza 'DEFINITIVA vs ESTRATO'**
df5 = df.groupby('ESTRATO')['DEFINITIVA'].agg(['mean', 'std', 'median', 'count'])
df5['coef_var(%)'] = round((df5['std'] / df5['mean']) * 100, 1)
# Calculate the IC for each group instead of the whole dataset
df5['IC(μ - 95%)'] = df5.apply(lambda row: (row['mean'] - 1.96 * row['std'] / np.sqrt(row['count']), row['mean'] + 1.96 * row['std'] / np.sqrt(row['count'])), axis=1)
```



	mean	std	median	count	coef_var(%)	IC(μ) - 95%
ESTRATO						
I	2.693333	1.274736	2.30	15	47.3	(2.048227808804379, 3.338438857862288)
II	2.880769	0.880917	2.80	26	30.6	(2.5421555013495882, 3.219382960188874)
III	2.905556	1.076198	2.95	18	37.0	(2.4083773422915673, 3.4027337688195436)
IV	2.650000	0.828989	2.60	10	31.3	(2.136187496367703, 3.163812503632297)
V	3.280000	0.766159	3.30	5	23.4	(2.6084318053987365, 3.951568194601263)

▼ C. Descriptores numericos e intervalos de confianza 'DEFINITIVA vs ESTRATO'

```
# @title **C. Descriptores numericos e intervalos de confianza 'DEFINITIVA vs ESTRATO'**
df6 = df.groupby('ESTRATO')['DEFINITIVA'].agg(['mean', 'std', 'median', 'count'])
df6['coef_var(%)'] = round((df6['std'] / df6['mean']) * 100, 1)
# Calculate the IC for each group instead of the whole dataset
# Round each element of the tuple individually
df6['IC(μ) - 95%'] = df6.apply(lambda row: (round(row['mean'] - 1.96 * row['std'] / np.sqrt(row['count']), 2), round(row['mean'] + 1.96 * row['std'] / np.sqrt(row['count']), 2)), axis=1)
```



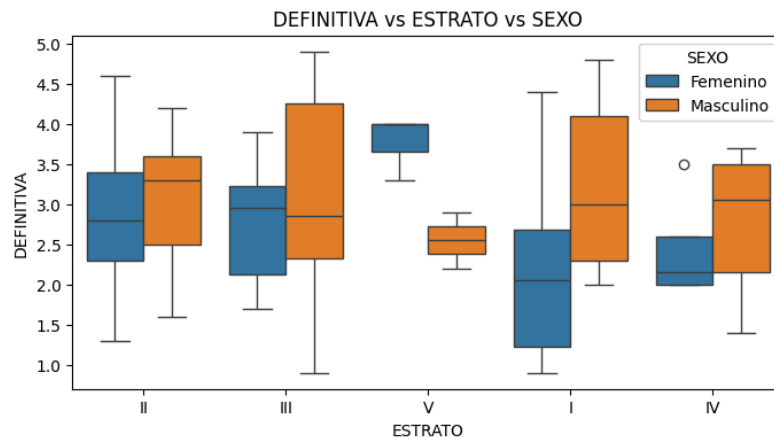
	mean	std	median	count	coef_var(%)	IC(μ) - 95%
ESTRATO						
I	2.693333	1.274736	2.30	15	47.3	(2.05, 3.34)
II	2.880769	0.880917	2.80	26	30.6	(2.54, 3.22)
III	2.905556	1.076198	2.95	18	37.0	(2.41, 3.4)
IV	2.650000	0.828989	2.60	10	31.3	(2.14, 3.16)
V	3.280000	0.766159	3.30	5	23.4	(2.61, 3.95)

[Volver al inicio](#)

11. Datos Multivariados - DEFINITIVA vs ESTRATO vs SEXO

A. BoxPlot para 'DEFINITIVA vs ESTRATO vs SEXO'

```
# @title **A. BoxPlot para 'DEFINITIVA vs ESTRATO vs SEXO'**
fig, ax = plt.subplots(1, 1, figsize=(8, 4))
ax.set_title('DEFINITIVA vs ESTRATO vs SEXO')
sns.boxplot(x="ESTRATO", y="DEFINITIVA", hue='SEXO', data=df, ax=ax);
```



B. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO'

```
# @title **B. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO'**
print('DEFINITIVA vs ESTRATO vs SEXO')
df.groupby(['ESTRATO', 'SEXO'])['DEFINITIVA'].agg(['count', 'min', 'max', 'mean', 'median', 'std'])
```




DEFINITIVA vs ESTRATO vs SEXO

ESTRATO	SEXO	count	min	max	mean	median	std
I	Femenino	8	0.9	4.4	2.225000	2.05	1.282576
	Masculino	7	2.0	4.8	3.228571	3.00	1.116116
II	Femenino	17	1.3	4.6	2.782353	2.80	0.909145
	Masculino	9	1.6	4.2	3.066667	3.30	0.844097
III	Femenino	10	1.7	3.9	2.810000	2.95	0.786624
	Masculino	8	0.9	4.9	3.025000	2.85	1.409914
IV	Femenino	4	2.0	3.5	2.450000	2.15	0.714143
	Masculino	6	1.4	3.7	2.783333	3.05	0.936839
V	Femenino	3	3.3	4.0	3.766667	4.00	0.404145
	Masculino	2	2.2	2.9	2.550000	2.55	0.494975

C. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO'

```
# @title **C. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO'**
df7 = df.groupby(['ESTRATO', 'SEXO'])['DEFINITIVA'].agg(['count', 'min', 'max', 'mean', 'median', 'std'])
df7['coef_var(%)'] = round((df7['std'] / df7['mean']) * 100, 1)
# Calculate the IC for each group instead of the whole dataset
# Round each element of the tuple individually
df7['IC( $\mu$ ) - 95%'] = df7.apply(lambda row: (round(row['mean'] - 1.96 * row['std'] / np.sqrt(row['count']), 2), round(row['mean'] + 1.96 * row['std'] / np.sqrt(row['count']), 2)), axis=1)
```

df7



		count	min	max	mean	median	std	coef_var(%)	IC(μ) - 95%
ESTRATO	SEXO								
I	Femenino	8	0.9	4.4	2.225000	2.05	1.282576	57.6	(1.34, 3.11)
	Masculino	7	2.0	4.8	3.228571	3.00	1.116116	34.6	(2.4, 4.06)
II	Femenino	17	1.3	4.6	2.782353	2.80	0.909145	32.7	(2.35, 3.21)
	Masculino	9	1.6	4.2	3.066667	3.30	0.844097	27.5	(2.52, 3.62)
III	Femenino	10	1.7	3.9	2.810000	2.95	0.786624	28.0	(2.32, 3.3)
	Masculino	8	0.9	4.9	3.025000	2.85	1.409914	46.6	(2.05, 4.0)
IV	Femenino	4	2.0	3.5	2.450000	2.15	0.714143	29.1	(1.75, 3.15)
	Masculino	6	1.4	3.7	2.783333	3.05	0.936839	33.7	(2.03, 3.53)
V	Femenino	3	3.3	4.0	3.766667	4.00	0.404145	10.7	(3.31, 4.22)
	Masculino	2	2.2	2.9	2.550000	2.55	0.494975	19.4	(1.86, 3.24)

[Volver al inicio](#)

12. Intervalos de Confianza para la Media usando Z y *t Student*


A. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO'

```
# @title **A. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO**
alpha = 0.05
df8 = df.groupby(['ESTRATO', 'SEXO'])['DEFINITIVA'].agg(['count', 'min', 'max', 'mean', 'median', 'std'])
df8['coef_var(%)'] = round((df8['std'] / df8['mean']) * 100,1)

# Asumiendo que 'count' representa el tamaño de la muestra para cada fila
df8['df'] = df8['count'] - 1 # Grados de libertad
df8['t - critico'] = stats.t.ppf(1 - alpha/2, df8['df']) # Grados de libertad

# Aplicamos la fórmula del IC con t de Student
df8['IC( $\mu$ )_t - 95%'] = df8.apply(lambda row: (
    round(row['mean'] - stats.t.ppf(1 - alpha/2, row['df']) * row['std'] / np.sqrt(row['count'])), 2),
    round(row['mean'] + stats.t.ppf(1 - alpha/2, row['df']) * row['std'] / np.sqrt(row['count'])), 2)
), axis=1)

df8
```



		count	min	max	mean	median	std	coef_var(%)	df	t - critico	IC(μ)_t - 95%
ESTRATO	SEXO										
I	Femenino	8	0.9	4.4	2.225000	2.05	1.282576	57.6	7	2.364624	(1.15, 3.3)
	Masculino	7	2.0	4.8	3.228571	3.00	1.116116	34.6	6	2.446912	(2.2, 4.26)
II	Femenino	17	1.3	4.6	2.782353	2.80	0.909145	32.7	16	2.119905	(2.31, 3.25)
	Masculino	9	1.6	4.2	3.066667	3.30	0.844097	27.5	8	2.306004	(2.42, 3.72)
III	Femenino	10	1.7	3.9	2.810000	2.95	0.786624	28.0	9	2.262157	(2.25, 3.37)
	Masculino	8	0.9	4.9	3.025000	2.85	1.409914	46.6	7	2.364624	(1.85, 4.2)
IV	Femenino	4	2.0	3.5	2.450000	2.15	0.714143	29.1	3	3.182446	(1.31, 3.59)
	Masculino	6	1.4	3.7	2.783333	3.05	0.936839	33.7	5	2.570582	(1.8, 3.77)
V	Femenino	3	3.3	4.0	3.766667	4.00	0.404145	10.7	2	4.302653	(2.76, 4.77)
	Masculino	2	2.2	2.9	2.550000	2.55	0.494975	19.4	1	12.706205	(-1.9, 7.0)

B. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO'

```
# @title **B. Descriptores Numéricos para 'DEFINITIVA vs ESTRATO vs SEXO**
alpha = 0.02

df9 = df.groupby(['ESTRATO', 'SEXO'])['DEFINITIVA'].agg(['count', 'min', 'max', 'mean', 'median', 'std'])
df9['coef_var(%)'] = round((df9['std'] / df9['mean']) * 100,1)

# Asumiendo que 'count' representa el tamaño de la muestra para cada fila
df9['df'] = df9['count'] - 1 # Grados de libertad
df9['t - critico'] = stats.t.ppf(1 - alpha/2, df9['df']) # Grados de libertad
df9['z - critico'] = stats.norm.ppf(1 - alpha/2) # Grados de libertad

# Aplicamos la fórmula del IC con t de Student
df9['f'IC( $\mu$ )_t - {1-alpha/2}%'] = df9.apply(lambda row: (
    round(row['mean'] - stats.t.ppf(1 - alpha/2, row['df']) * row['std'] / np.sqrt(row['count'])), 2),
    round(row['mean'] + stats.t.ppf(1 - alpha/2, row['df']) * row['std'] / np.sqrt(row['count'])), 2)
), axis=1)
```

```
), axis=1)
df9[f'IC(μ)_Z - {1-alpha/2}%'] = df9.apply(lambda row: (
    round(row['mean'] - stats.norm.ppf(1 - alpha/2) * row['std'] / np.sqrt(row['count'])), 2),
    round(row['mean'] + stats.norm.ppf(1 - alpha/2) * row['std'] / np.sqrt(row['count'])), 2)
), axis=1)

df9
```



		count	min	max	mean	median	std	coef_var(%)	df	t - critico	z - critico	IC(μ)_t -0.99%	IC(μ)_Z - 0.99%
ESTRATO	SEXO												
I	Femenino	8	0.9	4.4	2.225000	2.05	1.282576	57.6	7	2.997952	2.326348	(0.87, 3.58)	(1.17, 3.28)
	Masculino	7	2.0	4.8	3.228571	3.00	1.116116	34.6	6	3.142668	2.326348	(1.9, 4.55)	(2.25, 4.21)
II	Femenino	17	1.3	4.6	2.782353	2.80	0.909145	32.7	16	2.583487	2.326348	(2.21, 3.35)	(2.27, 3.3)
	Masculino	9	1.6	4.2	3.066667	3.30	0.844097	27.5	8	2.896459	2.326348	(2.25, 3.88)	(2.41, 3.72)
III	Femenino	10	1.7	3.9	2.810000	2.95	0.786624	28.0	9	2.821438	2.326348	(2.11, 3.51)	(2.23, 3.39)
	Masculino	8	0.9	4.9	3.025000	2.85	1.409914	46.6	7	2.997952	2.326348	(1.53, 4.52)	(1.87, 4.18)
IV	Femenino	4	2.0	3.5	2.450000	2.15	0.714143	29.1	3	4.540703	2.326348	(0.83, 4.07)	(1.62, 3.28)
	Masculino	6	1.4	3.7	2.783333	3.05	0.936839	33.7	5	3.364930	2.326348	(1.5, 4.07)	(1.89, 3.67)
V	Femenino	3	3.3	4.0	3.766667	4.00	0.404145	10.7	2	6.964557	2.326348	(2.14, 5.39)	(3.22, 4.31)
	Masculino	2	2.2	2.9	2.550000	2.55	0.494975	19.4	1	31.820516	2.326348	(-8.59, 13.69)	(1.74, 3.36)

[1 Volver al inicio](#)

13. Salvando la Base de Datos a Exportar nuestro Dataframes a limpios

Salvando la Base de Datos a Exportar nuestro Dataframes a limpios

```
# @title **Salvando la Base de Datos a Exportar nuestro Dataframes a limpios**
df.to_csv("df2.csv", index=False)
```

[1 Volver al inicio](#)

14. Rellamando los datos

Rellamando los datos

```
# @title **Rellamando los datos**
df2 = pd.read_csv("/content/df2.csv")
df2
```



	CURSO	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	NRC	PROGRAMA	EDAD	PESO	ESTATURA	...	B: Memoria y atención	C: Relacionar tu experiencias con lo que aprendes	D: Autoestima	E: Actitud hacia el Aprendizaje	An Fe
0	PROBABILIDAD	100	90	3.6	4.30	2314	F_NEGOCIOS	20	55	160	...	ALTO	MEDIO	ALTO	ALTO	
1	ESTADISTICA	70	75	0.9	2.50	1136	DERECHO	18	80	185	...	ALTO	ALTO	ALTO	ALTO	
2	PROBABILIDAD	85	95	3.9	3.80	2314	F_NEGOCIOS	19	60	158	...	BAJO	BAJO	BAJO	MEDIO	
3	PROBABILIDAD	5	5	2.9	0.50	2314	MECANICA	18	72	181	...	ALTO	BAJO	MEDIO	MEDIO	
4	ESTADISTICA	20	70	3.7	0.55	1009	PSICOLOGÍA	19	45	163	...	MEDIO	MEDIO	ALTO	ALTO	
...
69	PROBABILIDAD	90	90	2.3	2.10	2314	F_NEGOCIOS	18	59	176	...	MEDIO	MEDIO	ALTO	ALTO	
70	PROBABILIDAD	85	95	2.0	3.10	2314	F_NEGOCIOS	18	60	171	...	BAJO	ALTO	ALTO	ALTO	
71	ESTADISTICA	65	75	1.7	2.95	1136	DERECHO	20	55	164	...	MEDIO	MEDIO	ALTO	ALTO	
72	ESTADISTICA	100	100	2.3	3.20	2313	PSICOLOGÍA	19	67	171	...	BAJO	BAJO	BAJO	MEDIO	
73	ESTADISTICA	100	100	3.8	3.10	1136	DERECHO	18	60	165	...	ALTO	BAJO	ALTO	ALTO	

74 rows × 27 columns

[1 Volver al inicio](#)


15. Regresión Lineal Simple

A. Nuestras librerías mas usadas

```
# @title **A. Nuestras librerías mas usadas**
import numpy as np
from numpy.linalg import inv
import pandas as pd # para manejar los datos
import matplotlib.pyplot as plt # Para visualizar los datos
import random
import seaborn as sns
from sklearn.metrics import r2_score
```

B. Vamos hallar la Matriz de Correlaciones de nuestra data

```
# @title **B. Vamos hallar la Matriz de Correlaciones de nuestra data**
# Seleccionar solo las columnas numéricas para calcular la correlación
df3 = df2.select_dtypes(include=np.number)
df3
```




	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	NRC	EDAD	PESO	ESTATURA	DEFINITIVA
0	100	90	3.6	4.30	2314	20	55	160	3.6
1	70	75	0.9	2.50	1136	18	80	185	0.9
2	85	95	3.9	3.80	2314	19	60	158	3.9
3	5	5	2.9	0.50	2314	18	72	181	2.9
4	20	70	3.7	0.55	1009	19	45	163	3.7
...
69	90	90	2.3	2.10	2314	18	59	176	2.3
70	85	95	2.0	3.10	2314	18	60	171	2.0
71	65	75	1.7	2.95	1136	20	55	164	1.7
72	100	100	2.3	3.20	2313	19	67	171	2.3
73	100	100	3.8	3.10	1136	18	60	165	3.8

74 rows × 9 columns

C. Vamos eliminar la columna "NRC"

```
# @title **C. Vamos eliminar la columna "NRC"***
df3 = df3.drop(columns=['NRC'])
df3
```



	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	EDAD	PESO	ESTATURA	DEFINITIVA
0	100	90	3.6	4.30	20	55	160	3.6
1	70	75	0.9	2.50	18	80	185	0.9
2	85	95	3.9	3.80	19	60	158	3.9
3	5	5	2.9	0.50	18	72	181	2.9
4	20	70	3.7	0.55	19	45	163	3.7
...
69	90	90	2.3	2.10	18	59	176	2.3
70	85	95	2.0	3.10	18	60	171	2.0
71	65	75	1.7	2.95	20	55	164	1.7
72	100	100	2.3	3.20	19	67	171	2.3
73	100	100	3.8	3.10	18	60	165	3.8


74 rows × 8 columns

 [Volver al inicio](#)



16. Matriz de correlación

A. Calcular la matriz de correlación en el DataFrame numérico

```
# @title **A. Calcular la matriz de correlación en el DataFrame numérico**
df3.corr().round(2)
```



	ASISTENCIA2	ASISTENCIA1	PARCIAL 1	PARCIAL 2	EDAD	PESO	ESTATURA	DEFINITIVA
ASISTENCIA2	1.00	0.63	0.08	0.73	0.04	-0.06	-0.04	0.08
ASISTENCIA1	0.63	1.00	0.10	0.49	0.05	-0.04	-0.13	0.10
PARCIAL 1	0.08	0.10	1.00	0.35	-0.20	0.01	-0.11	1.00
PARCIAL 2	0.73	0.49	0.35	1.00	-0.11	-0.03	-0.07	0.35
EDAD	0.04	0.05	-0.20	-0.11	1.00	0.06	-0.08	-0.20
PESO	-0.06	-0.04	0.01	-0.03	0.06	1.00	0.66	0.01
ESTATURA	-0.04	-0.13	-0.11	-0.07	-0.08	0.66	1.00	-0.11
DEFINITIVA	0.08	0.10	1.00	0.35	-0.20	0.01	-0.11	1.00



▼ B. Lista los pares de variable con mayor correlacion


```
# @title **B. Lista los pares de variable con mayor correlacion**

correlation_matrix = df3.corr().abs()
# Eliminar la diagonal (correlación de una variable consigo misma)
np.fill_diagonal(correlation_matrix.values, 0)

# Desapilar la matriz para obtener pares de variables y sus correlaciones
stacked_corr = correlation_matrix.stack()

# Ordenar por correlación descendente
sorted_corr = stacked_corr.sort_values(ascending=False)

# Mostrar los pares con mayor correlación (ej. top 10)
print("Pares de variables con mayor correlación:")
print(sorted_corr.head(10))
```



Pares de variables con mayor correlación:

PARCIAL 1	DEFINITIVA	1.000000
DEFINITIVA	PARCIAL 1	1.000000
PARCIAL 2	ASISTENCIA2	0.726096
ASISTENCIA2	PARCIAL 2	0.726096
PESO	ESTATURA	0.658049
ESTATURA	PESO	0.658049
ASISTENCIA1	ASISTENCIA2	0.627203
ASISTENCIA2	ASISTENCIA1	0.627203
ASISTENCIA1	PARCIAL 2	0.489714
PARCIAL 2	ASISTENCIA1	0.489714

dtype: float64

 [Volver al inicio](#)

▼ 17. Metodos de Regresión Lineal - optimize.curve_fit

▼ A. Empecemos la Regresión Lineal - optimize.curve_fit

```
# @title **A. Empecemos la Regresión Lineal - `optimize.curve_fit`**
datax = df3['ESTATURA']
datay = df3['PESO']

def f( x, p0, p1):
    return p0 + p1*x

def ff(x, p):
    return f(x, *p)

# Estos son los verdaderos parámetros
p0 = 1.0
p1 = 1.0

# Estas son conjeturas iniciales para ajustes:
pstart = [
    p0 + random.random(),
    p1 + random.random(),
]
```

▼ B. Optimizar la Curva de ajuste - optimize.curve_fit

```
# @title **B. Optimizar la Curva de ajuste - `optimize.curve_fit`**
from scipy import optimize
err_stdev = 0.2
def fit_curvefit(p0, datax, datay, function, yerr=err_stdev, **kwargs):# Definimos los parametros de nuestra función
```



```

"""
Nota: Según la documentación actual (Scipy V1.1.0), sigma (yerr) debe ser:
Ninguno o secuencia de longitud M o matriz MxM, opcional
Por lo tanto, reemplace:
    err_stdev = 0.2
Con:
    err_stdev = [0.2 para elemento en xdata]
O similar, para crear una secuencia de longitud M para este ejemplo.
"""
pfit3, pcov = optimize.curve_fit(f, datax, datay, p0=p0, sigma=None, epsfcn=0.0001, **kwargs) # usamos el metodo de curva fit

# ajustamos los datos y residuos
error = []
for i in range(len(pfit3)):
    try:
        error.append(np.absolute(pcov[i][i])**0.5)
    except:
        error.append( 0.00 )
pfit_curvefit = pfit3
perr_curvefit = np.array(error)

return pfit_curvefit, perr_curvefit
print('Algunas visualizaciones de lo construido hasta ahora:')
print('Lo almacenado en pfit_curvefit es ', pfit_curvefit)
print('Lo almacenado en perr_curvefit =', perr_curvefit)
pfit3, perr = fit_curvefit(pstart, datax, datay, ff)

# @title **B. Optimizar la Curva de ajuste - `optimize.curve_fit`**
from scipy import optimize
err_stdev = 0.2
def fit_curvefit(p0, datax, datay, function, yerr=err_stdev, **kwargs):# Definimos los parametros de nuestra función
"""
Nota: Según la documentación actual (Scipy V1.1.0), sigma (yerr) debe ser:
Ninguno o secuencia de longitud M o matriz MxM, opcional
Por lo tanto, reemplace:
    err_stdev = 0.2
Con:
    err_stdev = [0.2 para elemento en xdata]
O similar, para crear una secuencia de longitud M para este ejemplo.
"""
pfit3, pcov = optimize.curve_fit(f, datax, datay, p0=p0, sigma=None, epsfcn=0.0001, **kwargs) # usamos el metodo de curva fit

# ajustamos los datos y residuos
error = []
for i in range(len(pfit3)):
    try:
        error.append(np.absolute(pcov[i][i])**0.5)
    except:
        error.append( 0.00 )
pfit_curvefit = pfit3
perr_curvefit = np.array(error)

return pfit_curvefit, perr_curvefit
print('Algunas visualizaciones de lo construido hasta ahora:')
print('Lo almacenado en pfit_curvefit es ', pfit_curvefit)
print('Lo almacenado en perr_curvefit =', perr_curvefit)
pfit3, perr = fit_curvefit(pstart, datax, datay, ff)

print("\n# Fit parameters and parameter errors from curve_fit method :)")
print("pfit = ", pfit3)
print("perr = ", perr)

```



```

# Fit parameters and parameter errors from curve_fit method :
pfit = [-84.12672712  0.87564223]
perr = [19.90917368  0.11808172]

```

✓ C. Bondad de Ajuste R^2 para el método de regresión `curve_fit`

```

# @title **C. Bondad de Ajuste  $R^2$  para el método de regresión `curve_fit`**
from sklearn.metrics import r2_score
r2 = r2_score(datay, f( datax, *pfit3))
print('El coeficiente de Bondad de Ajuste o de Determinación del modelo R_cuadrado = ', round(r2,3))
print('Esto es, el modelo explica la variabilidad observada en la respuesta en un porcentaje de ', round(r2*100,1), '%')

```



```

El coeficiente de Bondad de Ajuste o de Determinación del modelo R_cuadrado = 0.433
Esto es, el modelo explica la variabilidad observada en la respuesta en un porcentaje de 43.3 %

```

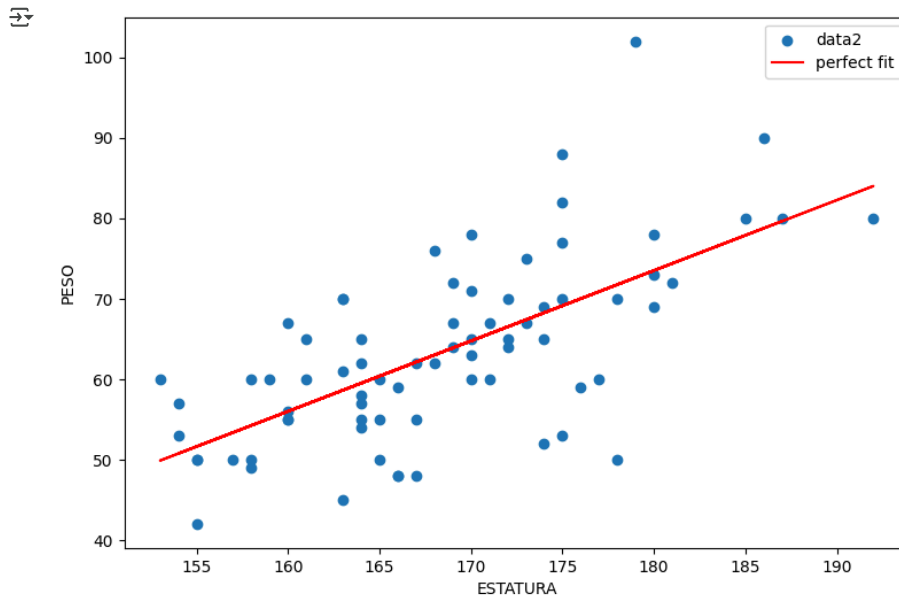
✓ D. Diagrama de Dispersión y curva `perfect fit`

```

# @title **D. Diagrama de Dispersión y curva `perfect fit`**
plt.figure(figsize=(30,6))
plt.subplot(131)
plt.scatter(datax, datay, label="data2")
x0=datax #np.linspace(data["G3"].min(),data["G3"].max(),7)
plt.plot(x0,f(x0,*pfit3),color="r",label="perfect fit")

```

```
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.legend()
plt.show()
```



[Volver al inicio](#)

18. Regresión lineal - MINIMOS CUADRADOS - Least Squares

A. Regresión lineal - MINIMOS CUADRADOS - Least Squares

```
# @title **A. Regresión lineal - MINIMOS CUADRADOS - Least Squares**
import statsmodels.api as sm
from statsmodels.tools.eval_measures import rmse
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

B. Regresión lineal - MINIMOS CUADRADOS - OLS

```
# @title **B. Regresión lineal - MINIMOS CUADRADOS - OLS**
# nuestras variables independiente y dependiente

X = df3['ESTATURA']
y = df3['PESO']

# para obtener intercepción -- esto es opcional

X = sm.add_constant(X)

# ajustar el modelo de regresión

reg = sm.OLS(y, X).fit()
reg.summary()
```



OLS Regression Results

Dep. Variable: PESO **R-squared:** 0.433
Model: OLS **Adj. R-squared:** 0.425
Method: Least Squares **F-statistic:** 54.99
Date: Wed, 11 Jun 2025 **Prob (F-statistic):** 1.88e-10
Time: 18:13:05 **Log-Likelihood:** -263.30
No. Observations: 74 **AIC:** 530.6
Df Residuals: 72 **BIC:** 535.2
Df Model: 1
Covariance Type: nonrobust

	coef	std err	t	P> t	[0.025	0.975]
const	-84.1267	19.909	-4.226	0.000	-123.815	-44.439
ESTATURA	0.8756	0.118	7.416	0.000	0.640	1.111

Omnibus: 5.164 **Durbin-Watson:** 2.163
Prob(Omnibus): 0.076 **Jarque-Bera (JB):** 5.494
Skew: 0.308 **Prob(JB):** 0.0641
Kurtosis: 4.184 **Cond. No.** 3.35e+03

Notes:

[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

[2] The condition number is large, 3.35e+03. This might indicate that there are strong multicollinearity or other numerical problems.

▼ C. Hacer el grafico de Dispersión pero usando Minimos cuadrados:

```

# @title **C. Hacer el grafico de Dispersión pero usando Minimos cuadrados:**

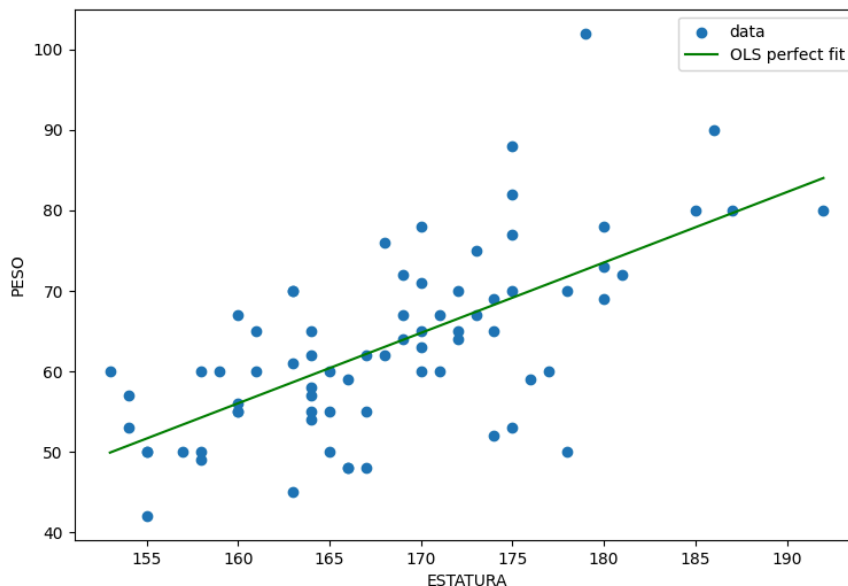
# Plotting using the OLS results from Paso 13
plt.figure(figsize=(30,6))
plt.subplot(131)

plt.scatter(datax, datay, label="data")

p0_ols = reg.params['const']
p1_ols = reg.params['ESTATURA']
# Define the function for the OLS line
def ols_line(x, p0, p1):
    return p0 + p1 * x

x_plot = np.linspace(datax.min(), datax.max(), 100)
plt.plot(x_plot, ols_line(x_plot, p0_ols, p1_ols), color="g", label="OLS perfect fit")
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.legend()
plt.show()

```



[Volver al inicio](#)

▼ 19. Regresión lineal - Machine Learning

▼ A. importando `train_test_split` desde `sklearn`

```
# @title **A. importando `train_test_split` desde `sklearn`**
from sklearn.model_selection import train_test_split

# nuestras variables independiente y dependiente
X = df3['ESTATURA']
y = df3['PESO']

# para obtener intercepción -- esto es opcional

X = sm.add_constant(X)

# dividir los datos en entrenamiento y prueba

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

▼ B. Selección del método de Regresión Lineal de la Biblioteca `scikit-learn`

```
# @title **B. Selección del método de Regresión Lineal de la Biblioteca scikit-learn**

from sklearn.linear_model import LinearRegression

# creando un objeto de la clase LinearRegression

LR = LinearRegression()

# ajustando los datos de entrenamiento

LR.fit(x_train,y_train)

# modelo de regresión
modelo = LR.fit(x_train,y_train)
```

▼ C. Evaluación del modelo entrenado sobre los datos de entrenamiento

```
# @title **C. Evaluación del modelo entrenado sobre los datos de entrenamiento**

from sklearn import metrics
from sklearn import metrics # Import the metrics module

y_prediction_train = modelo.predict(x_train)
print('Error Absoluto Medio en datos train es MAE =', metrics.mean_absolute_error(y_train,y_prediction_train))
y_prediction_train
y_prediction_train =np.array(y_prediction_train)
df_train = pd.DataFrame(y_prediction_train)
df_train
```

Error Absoluto Medio en datos train es MAE = 6.6489264081670205

0

0 61.312077
1 71.308419
2 72.217177
3 60.403318
4 67.673385
5 65.855868
6 55.859526
7 59.494560
8 68.582144
9 79.487244
10 51.315734
11 66.764627
12 55.859526
13 61.312077
14 51.315734
15 59.494560
16 74.034694
17 54.950768
18 69.490902
19 51.315734
20 64.947110
21 69.490902
22 74.943452
23 66.764627
24 62.220835
25 56.768285
26 64.038352
27 53.133251
28 62.220835
29 64.947110

▼ D. Evaluación del modelo entrenado sobre los datos de prueba

30 74.034694
31 54.042010

```
# @title **D. Evaluación del modelo entrenado sobre los datos de prueba**
y_prediction_test = modelo.predict(x_test)
print('Error Absoluto Medio en datos de prueba es MAE =', metrics.mean_absolute_error(y_test,y_prediction_test))
y_prediction_test
y_prediction_test =np.array(y_prediction_test)
df_test = pd.DataFrame(y_prediction_test)
df_test
```

30 69.490902
37 63.129593
38 72.217177
39 64.947110
40 50.406976
41 59.494560
42 69.490902
43 70.399660
44 66.764627
45 50.406976
46 64.947110
47 58.585801
48 58.585801
49 78.578486
50 59.494560
51 67.673385
52 54.042010

↗ Error Cuadrático Medio en datos de prueba es MAE = 4.72105456332543

```

0 58.585801
1 58.585801
2 54.042010
3 55.859526
4 74.034694
5 60.403318
6 68.582144
7 84.939794
8 49.498218
9 80.396002
10 62.220835
11 55.859526
12 56.768285
13 63.129593
14 64.038352

```

▼ E. Importando r2_score module

```

# @title **E. Importando r2_score module**
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
# predicting the accuracy score
score=r2_score(y_test,y_prediction_test)
print("El valor de R2 =",score)
print("Error Cuadrático Medio MSE ==",mean_squared_error(y_test,y_prediction_test))
print("Raiz Error Cuadrático Medio RMSE ==",np.sqrt(mean_squared_error(y_test,y_prediction_test)))

```

↗ El valor de R2 = 0.5891389772870637
 Error Cuadrático Medio MSE == 48.792029008398494
 Raiz Error Cuadrático Medio RMSE == 6.98512913326579

▼ F. Predicción total

```

# @title **F. Predicción total**
y_prediction_all = LR.predict(X)
print(pd.DataFrame(np.array(y_prediction_all)))

```

↗

```

0 55.859526
1 78.578486
2 54.042010
3 74.943452
4 58.585801
.. ..
69 70.399660
70 65.855868
71 59.494560
72 65.855868
73 60.403318

[74 rows x 1 columns]

```

▼ F. Hacer el grafico de Dispersión pero usando Machine Learning obtenido

```

# @title **F. Hacer el grafico de Dispersión pero usando Machine Learning obtenido**

# Plotting using the Machine Learning results from Paso 18
plt.figure(figsize=(30,6))
plt.subplot(131)

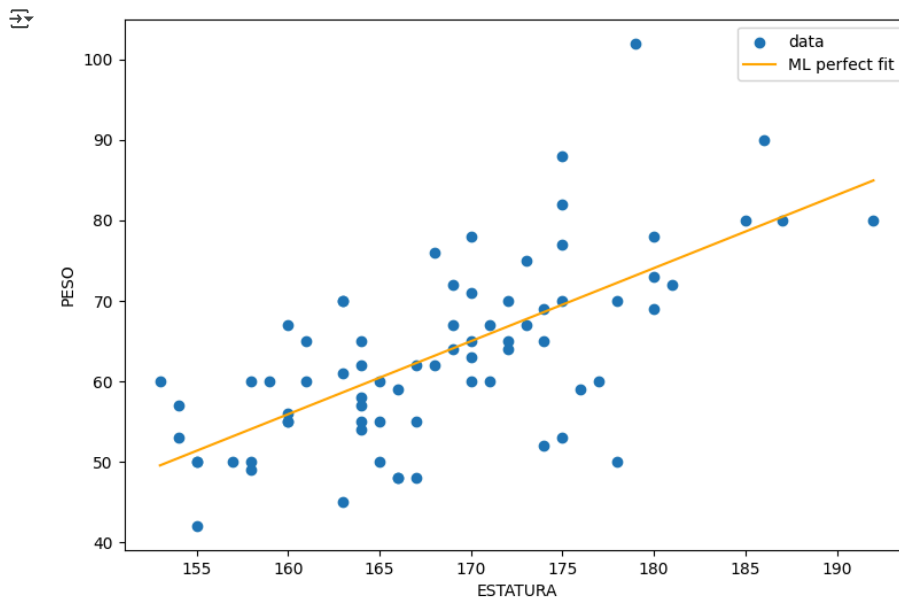
plt.scatter(datax, datay, label="data")

# Get the predicted values for all data points using the trained ML model
y_prediction_all = LR.predict(X)

# Sort the data by the independent variable (ESTATURA) to ensure the line is plotted correctly
sorted_indices = np.argsort(datax)
datax_sorted = datax.iloc[sorted_indices]
y_prediction_all_sorted = y_prediction_all[sorted_indices]

```

```
plt.plot(datax_sorted, y_prediction_all_sorted, color="orange", label="ML perfect fit")
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.legend()
plt.show()
```



[Volver al inicio](#)

20. Regresión Lineal con Bootstrap y Diagrama de Dispersión

Paso A. Regresión Lineal con Bootstrap y Diagrama de Dispersión

```
# @title **Paso A. Regresión Lineal con Bootstrap y Diagrama de Dispersión**

# Definir la función para la regresión lineal
def linear_model(x, a, b):
    return a + b * x

# Número de remuestreos bootstrap
n_bootstrap = 1000

# Listas para almacenar los coeficientes estimados de cada remuestreo
bootstrap_a = []
bootstrap_b = []

# Realizar remuestreo bootstrap
for _ in range(n_bootstrap):
    # Muestrear con reemplazo de los datos originales
    sample_indices = np.random.choice(len(datax), size=len(datax), replace=True)
    bootstrap_x = datax.iloc[sample_indices]
    bootstrap_y = datay.iloc[sample_indices]

    # Ajustar el modelo lineal a la muestra bootstrap
    try:
        popt, pcov = optimize.curve_fit(linear_model, bootstrap_x, bootstrap_y)
        bootstrap_a.append(popt[0])
        bootstrap_b.append(popt[1])
    except RuntimeError:
        # Manejar casos donde curve_fit no converge
        pass

# Calcular los coeficientes promedio de bootstrap
mean_a = np.mean(bootstrap_a)
mean_b = np.mean(bootstrap_b)

print(f"Coefficiente 'a' (Intercepto) promedio con Bootstrap: {mean_a:.4f}")
print(f"Coefficiente 'b' (Pendiente) promedio con Bootstrap: {mean_b:.4f}")

# Dibujar el diagrama de dispersión y la línea de regresión bootstrap
plt.figure(figsize=(8, 6))
plt.scatter(datax, datay, label="Datos", alpha=0.6)

# Dibujar la línea de regresión basada en los coeficientes promedio de bootstrap
x_plot = np.linspace(datax.min(), datax.max(), 100)
plt.plot(x_plot, linear_model(x_plot, mean_a, mean_b), color="purple", label="Regresión Bootstrap")
```

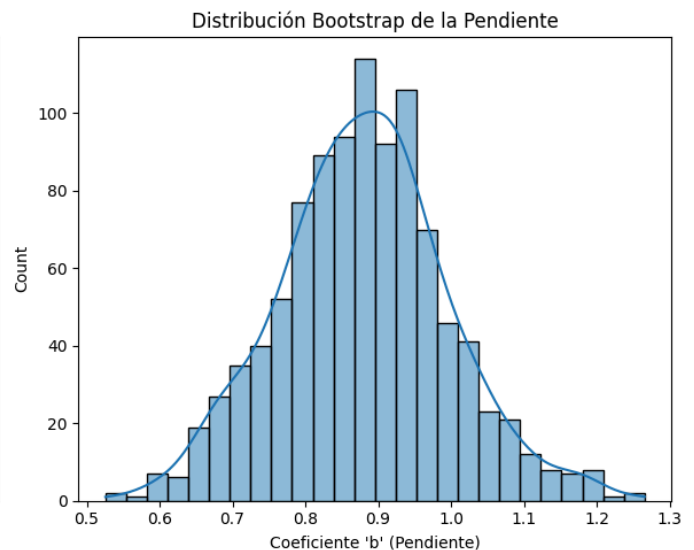
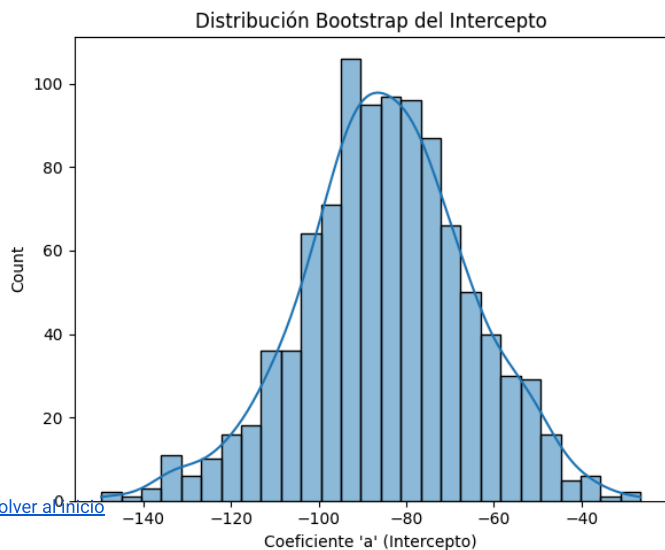
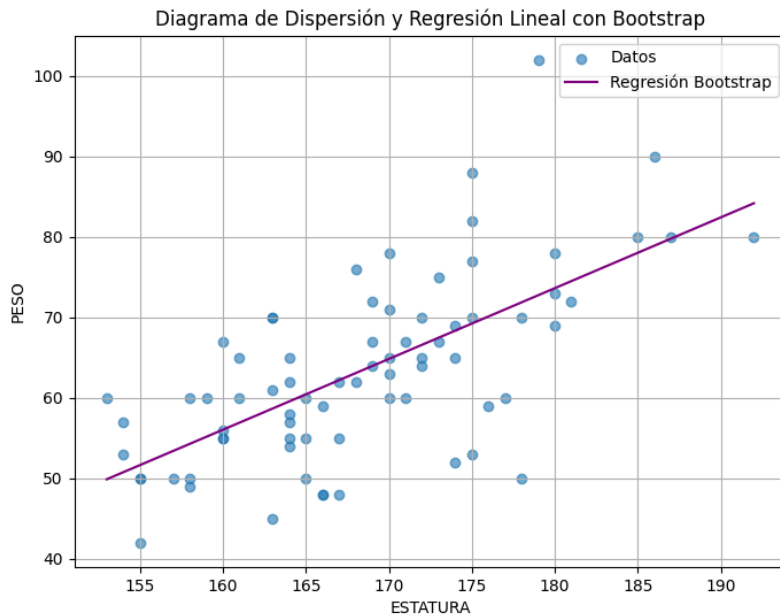
```
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.title("Diagrama de Dispersión y Regresión Lineal con Bootstrap")
plt.legend()
plt.grid(True)
plt.show()
```

```
# Opcional: Visualizar la distribución de los coeficientes bootstrap
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
sns.histplot(bootstrap_a, kde=True)
plt.xlabel("Coeficiente 'a' (Intercepto)")
plt.title("Distribución Bootstrap del Intercepto")
```

```
plt.subplot(1, 2, 2)
sns.histplot(bootstrap_b, kde=True)
plt.xlabel("Coeficiente 'b' (Pendiente)")
plt.title("Distribución Bootstrap de la Pendiente")
```

```
plt.tight_layout()
plt.show()
```

Coeficiente 'a' (Intercepto) promedio con Bootstrap: -84.6812
Coeficiente 'b' (Pendiente) promedio con Bootstrap: 0.8794



[Volver al inicio](#)

21. Un solo código para tres métodos de regresión lineal

A. Un solo código trabaja los tres métodos de regresión lineal estudiados con los gráficos

```
# @title **A. Un solo código trabaja los tres métodos de regresión lineal estudiados con los gráficos**
```

```
# Combinar los gráficos en una sola figura para comparación
```



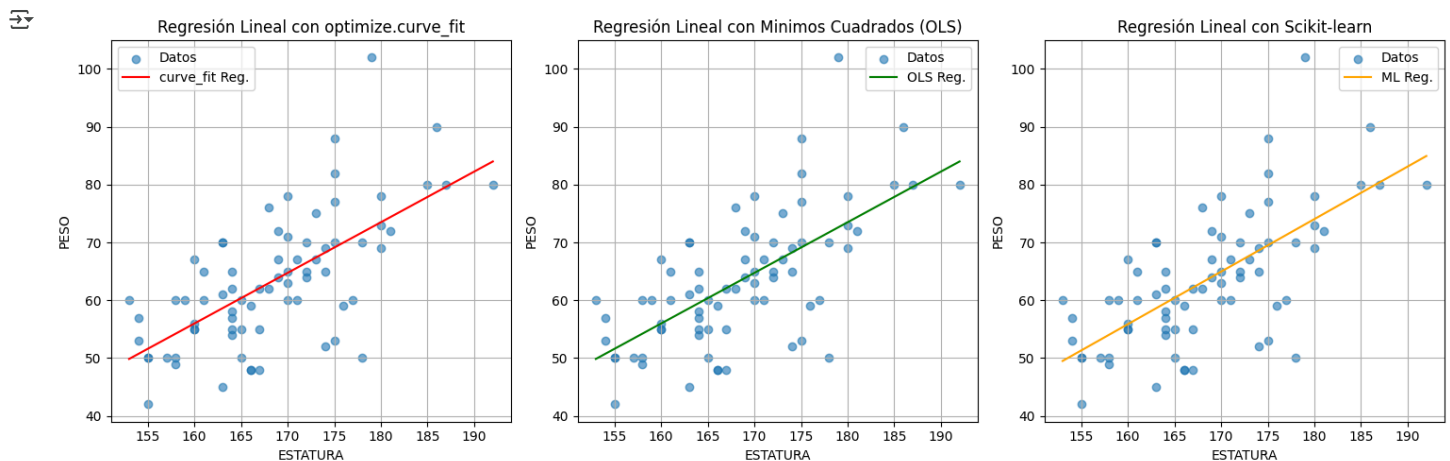
```
plt.figure(figsize=(15, 5))

# Gráfico para optimize.curve_fit (Regresión 1)
plt.subplot(1, 3, 1)
plt.scatter(datax, datay, label="Datos", alpha=0.6)
x0 = np.linspace(datax.min(), datax.max(), 100)
plt.plot(x0, f(x0, *pfit3), color="r", label="curve_fit Reg.")
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.title("Regresión Lineal con optimize.curve_fit")
plt.legend()
plt.grid(True)

# Gráfico para OLS (Regresión 2)
plt.subplot(1, 3, 2)
plt.scatter(datax, datay, label="Datos", alpha=0.6)
x_plot = np.linspace(datax.min(), datax.max(), 100)
plt.plot(x_plot, ols_line(x_plot, p0_ols, p1_ols), color="g", label="OLS Reg.")
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.title("Regresión Lineal con Mínimos Cuadrados (OLS)")
plt.legend()
plt.grid(True)

# Gráfico para Scikit-learn Linear Regression (Regresión 3)
plt.subplot(1, 3, 3)
plt.scatter(datax, datay, label="Datos", alpha=0.6)
# Get the predicted values for all data points using the trained ML model
y_prediction_all = LR.predict(X) # Use the full X data here
# Sort the data by the independent variable (ESTATURA) to ensure the line is plotted correctly
sorted_indices = np.argsort(datax)
datax_sorted = datax.iloc[sorted_indices]
y_prediction_all_sorted = y_prediction_all[sorted_indices]
plt.plot(datax_sorted, y_prediction_all_sorted, color="orange", label="ML Reg.")
plt.xlabel("ESTATURA")
plt.ylabel("PESO")
plt.title("Regresión Lineal con Scikit-learn")
plt.legend()
plt.grid(True)

plt.tight_layout() # Ajustar el diseño para evitar superposición
plt.show()
```



[Volver al inicio](#)

22. Los cuatro metodos de regresion vistos, los parametros calculados, AIC y el valor de R^2

A. Los cuatro metodos de regresion vistos, los parametros calculados, AIC y el valor de R^2

```
# @title **A. Los cuatro metodos de regresion vistos, los parametros calculados, AIC y el valor de  $R^2$ **
# nuestras variables independiente y dependiente

X_estatura = df3['ESTATURA'] # Usar un nombre diferente para la variable independiente en este paso
y_peso = df3['PESO'] # Usar un nombre diferente para la variable dependiente en este paso

# para obtener intercepción -- esto es opcional

X_estatura = sm.add_constant(X_estatura)

# ajustar el modelo de regresión y almacenar en una nueva variable
reg_estatura_peso = sm.OLS(y_peso, X_estatura).fit()
reg_estatura_peso.summary()
```

```

# Crear un diccionario para almacenar los resultados
results = {
    'Método': [],
    'Parámetro A (Intercepto)': [],
    'Parámetro B (Pendiente)': [],
    'Error Estándar A': [],
    'Error Estándar B': [],
    'AIC': [],
    'R2': []
}

# --- Método 1: optimize.curve_fit ---
# Los parámetros y sus errores ya se calcularon en pfit3 y perr
results['Método'].append('optimize.curve_fit')
results['Parámetro A (Intercepto)'].append(pfit3[0])
results['Parámetro B (Pendiente)'].append(pfit3[1])
results['Error Estándar A'].append(perr[0])
results['Error Estándar B'].append(perr[1])
# curve_fit no proporciona AIC directamente, se puede calcular pero requiere más pasos.
# Para simplificar, lo marcamos como no disponible.
results['AIC'].append('N/A')
results['R2'].append(r2_score(datay, f( datax, *pfit3))) # R2 calculado en Paso 10

# --- Método 2: Mínimos Cuadrados (statsmodels OLS) ---
# Los resultados del summary de OLS tienen los parámetros, errores estándar y AIC/R2
# Usar la variable reg_estatura_peso que contiene los resultados del OLS para Estatura vs Peso
results['Método'].append('Mínimos Cuadrados (OLS)')
results['Parámetro A (Intercepto)'].append(reg_estatura_peso.params['const'])
results['Parámetro B (Pendiente)'].append(reg_estatura_peso.params['ESTATURA'])
results['Error Estándar A'].append(reg_estatura_peso.bse['const'])
results['Error Estándar B'].append(reg_estatura_peso.bse['ESTATURA'])
results['AIC'].append(reg_estatura_peso.aic)
results['R2'].append(reg_estatura_peso.rsquared)

# --- Método 3: Machine Learning (scikit-learn LinearRegression) ---
# scikit-learn da los coeficientes y R2, pero no errores estándar directos ni AIC
# Ya se calculó en Pasos 18-21
# Ensure LR.coef_ is handled correctly for single feature case
results['Método'].append('Machine Learning (sklearn LR)')
# Access intercept
results['Parámetro A (Intercepto)'].append(LR.intercept_[0] if isinstance(LR.intercept_, np.ndarray) else LR.intercept_)
# Access coefficient for the single feature ('ESTATURA')
results['Parámetro B (Pendiente)'].append(LR.coef_[0][1] if isinstance(LR.coef_[0], np.ndarray) else (LR.coef_[1] if len(LR.coef_) > 1 else LR.coef_[0])) # Adjuste
results['Error Estándar A'].append('N/A') # scikit-learn no da errores estándar directos
results['Error Estándar B'].append('N/A') # scikit-learn no da errores estándar directos
results['AIC'].append('N/A') # scikit-learn no da AIC directo
results['R2'].append(r2_score(y_test, y_prediction_test)) # R2 calculado en Paso 21 (sobre datos de prueba)

# --- Método 4: Bootstrap ---
# Los coeficientes promedio de bootstrap se calcularon en Paso 25
# Los errores estándar se pueden estimar a partir de la desviación estándar de las distribuciones bootstrap
results['Método'].append('Bootstrap')
results['Parámetro A (Intercepto)'].append(mean_a)
results['Parámetro B (Pendiente)'].append(mean_b)
results['Error Estándar A'].append(np.std(bootstrap_a)) # Estimado del error estándar via bootstrap
results['Error Estándar B'].append(np.std(bootstrap_b)) # Estimado del error estándar via bootstrap
results['AIC'].append('N/A') # Bootstrap no da AIC
# Calcular R2 para el modelo con coeficientes promedio de bootstrap
r2_bootstrap = r2_score(datay, linear_model(datax, mean_a, mean_b))
results['R2'].append(r2_bootstrap)

# Crear el DataFrame a partir del diccionario
results_df = pd.DataFrame(results)

# Formatear para mejor lectura (opcional)
results_df = results_df.round(4)
results_df['AIC'] = results_df['AIC'].apply(lambda x: round(x, 2) if isinstance(x, (int, float)) else x)

# Mostrar la tabla
print("\nTabla Comparativa de Métodos de Regresión")
print(results_df.to_markdown(index=False))

```



Tabla Comparativa de Métodos de Regresión

Método	Parámetro A (Intercepto)	Parámetro B (Pendiente)	Error Estándar A	Error Estándar B	AIC	R2
optimize.curve_fit	-84.1267	0.8756	19.909173676590267	0.11808171777704271	N/A	0.433
Mínimos Cuadrados (OLS)	-84.1267	0.8756	19.90917367659037	0.1180817177770434	530.59	0.433
Machine Learning (sklearn LR)	-89.5418	0.9088	N/A	N/A	N/A	0.5891
Bootstrap	-84.6812	0.8794	19.2316464316367	0.11628643202211655	N/A	0.433

[Volver al inicio](#)

23. Muestreo de Medias y Varianzas de una Variable

▼ A. Muestreo de Medias y Varianzas de una Variable

```
# @title **A. Muestreo de Medias y Varianzas de una Variable**
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import random
n = 1000
a = 18 # Valor mínimo en un dado
b = 22 # Valor máximo en un dado
x=1 # nos va indicar el primer lanzamiento
E1 = [] # Edad del Estudiante 1
E2 = [] # Edad del Estudiante 2
E3 = [] # Edad del Estudiante 3
E4 = [] # Edad del Estudiante 4
E5 = [] # Edad del Estudiante 5

suma12 = [] # Edad Total de los cinco estudiantes
media = [] # Edad Promedio de los cinco estudiantes
varianza = [] # Varianza Muestral de la Edad de los cinco estudiantes
suma = 0 # Acumular los resultados sumandolos

while x<=n: # Esto es para decir que queremos lanzar el dado 10 veces
    valor1=random.randint(a,b) # Edad Estudiante 1
    E1.append(valor1) # Almacenar Edad Estudiante 1
    valor2=random.randint(a,b) # Edad Estudiante 2
    E2.append(valor2) # Almacenar Edad Estudiante 2
    valor3=random.randint(a,b) # Edad Estudiante 3
    E3.append(valor3) # Almacenar Edad Estudiante 3
    valor4=random.randint(a,b) # Edad Estudiante 4
    E4.append(valor4) # Almacenar Edad Estudiante 4
    valor5=random.randint(a,b) # Edad Estudiante 5
    E5.append(valor5) # Almacenar Edad Estudiante 5
    suma12.append(valor1+valor2+valor3+valor4+valor5)

    # Calculate the mean for the current iteration
    current_mean = (valor1+valor2+valor3+valor4+valor5)/5
    media.append(current_mean)

    # Use the current mean for variance calculation
    varianza.append((valor1**2+valor2**2+valor3**2+valor4**2+valor5**2-5*(current_mean**2))/4)

    suma=suma+valor1+valor2+valor3+valor4+valor5 # Acumulando la suma de los valores obtenidos

    x=x+1 # el ciclo se cierra y comienza
promedio=suma/n #Deseo hallar la media de los resultados de los 10 lanzamientos

EDAD = pd.DataFrame({'E1':E1,'E2':E2,'E3':E3,'E4':E4,'E5':E5,'Total':suma12,'media':media,'varianza':varianza})
EDAD.head()
```



	E1	E2	E3	E4	E5	Total	media	varianza
0	18	20	21	22	22	103	20.6	2.8
1	22	21	21	20	21	105	21.0	0.5
2	20	22	21	18	18	99	19.8	3.2
3	18	18	20	20	22	98	19.6	2.8
4	20	22	19	19	18	98	19.6	2.3

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

# Supongamos que 'dados' es tu DataFrame y ya tienes cargados los datos pertinentes

# Definir las columnas de interés
cols = EDAD.columns[5:]

# Número de columnas
N = len(cols)

# Crear figura y subplots
plt.figure(figsize=(25, 6))

# Para cada columna, ajustar una distribución normal y trazar la curva PDF
for i, col in enumerate(cols):
    plt.subplot(1, N, i+1)
    sns.histplot(EDAD[col], kde=True, stat="density", kde_kws=dict(cut=3),
                 alpha=.4, edgecolor=(1, 1, 1, .4))

# Ajustar una distribución normal a los datos
```

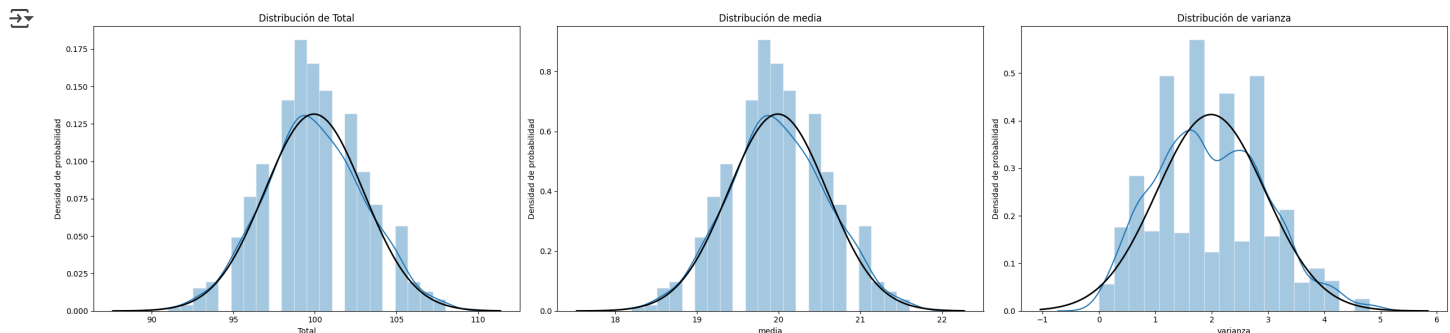
```
mu, std = norm.fit(EDAD[col])

# Calcular la PDF de la distribución normal ajustada
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 10000)
p = norm.pdf(x, mu, std)

# Trazar la curva de densidad de probabilidad (PDF)
plt.plot(x, p, 'k', linewidth=2)

# Configuración del gráfico
plt.title(f'Distribución de {col}')
plt.xlabel(col)
plt.ylabel('Densidad de probabilidad')

plt.tight_layout()
plt.show()
```



[Volver al inicio](#)

24. La simulacion es para EDAD de distribución de medias y varianzas

La simulacion es para EDAD de distribución de medias y varianzas

```
# @title **La simulacion es para EDAD de distribución de medias y varianzas**

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm
import matplotlib.animation as animation
from IPython.display import HTML

# Parámetros de la simulación
K = 100 # Total muestras
Tamaño_muestra = 5
# Rango de edad para la simulación
min_edad = df['EDAD'].min()
max_edad = df['EDAD'].max()

# Simulación de selección de edades de la columna EDAD
resultados = []
for _ in range(K):
    # Seleccionar con reemplazo del rango de edades observadas en la columna EDAD
    seleccion_edades = np.random.randint(min_edad, max_edad + 1, size=Tamaño_muestra)
    resultados.append(seleccion_edades)

# Convertir los resultados a un DataFrame de Pandas
df_sim_edad = pd.DataFrame(resultados, columns=[f'Edad{i+1}' for i in range(Tamaño_muestra)])
df_sim_edad['Suma'] = df_sim_edad.sum(axis=1)
df_sim_edad['Media'] = df_sim_edad.mean(axis=1)
df_sim_edad['Varianza'] = df_sim_edad.var(axis=1)

# Create the figure and axes outside the update function
fig, axs = plt.subplots(1, 3, figsize=(15, 5))

# Función para actualizar los gráficos en cada fotograma de la animación
def actualizar_sim_edad(i):
    axs[0].clear()
    axs[0].text(0.5, 0.5, f'Muestra {i+1}\nValores: {df_sim_edad.iloc[i, :Tamaño_muestra].tolist()}\nMedia: {df_sim_edad["Media"].iloc[i]:.2f}',
                fontsize=15, ha='center', va='center', transform=axs[0].transAxes)
    axs[0].axis('off')

    axs[1].clear()
    axs[1].hist(df_sim_edad['Media'][:i+1], bins=np.linspace(df_sim_edad['Media'].min(), df_sim_edad['Media'].max(), 10), color='skyblue', edgecolor='black')
    axs[1].set_title('Distribución Muestral de la Media (EDAD)')
    axs[1].set_xlabel('Media')
    axs[1].set_ylabel('Frecuencia')
    axs[1].set_xlim(df_sim_edad['Media'].min(), df_sim_edad['Media'].max())
```

```

axs[2].clear()
axs[2].hist(df_sim_edad['Varianza'][:i+1], bins=np.linspace(df_sim_edad['Varianza'].min(), df_sim_edad['Varianza'].max(), 8) if not df_sim_edad['Varianza'].isnull().all():
axs[2].set_title('Distribución Muestral de la Varianza (EDAD)')
axs[2].set_xlabel('Varianza')
axs[2].set_ylabel('Frecuencia')
if not df_sim_edad['Varianza'].isnull().all():
    axs[2].set_xlim(df_sim_edad['Varianza'].min(), df_sim_edad['Varianza'].max())

plt.tight_layout()

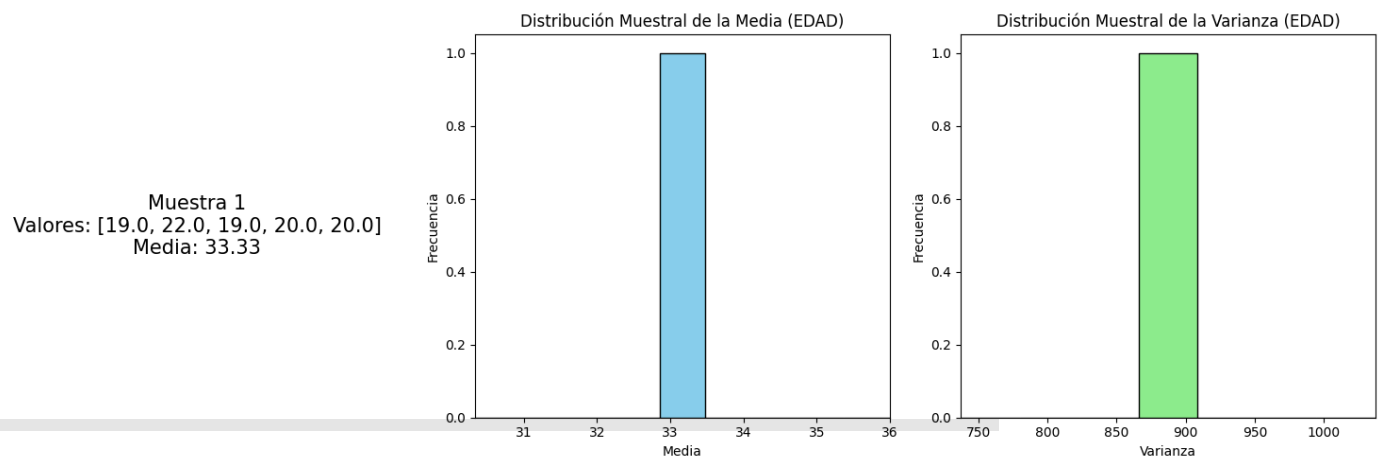
# Configuración de la animación
ani_sim_edad = animation.FuncAnimation(fig, actualizar_sim_edad, frames=K, interval=500, repeat=False)

# Mostrar la animación en el notebook
HTML(ani_sim_edad.to_html5_video())

```



0:00 / 0:50


[Volver al inicio](#)

✓ *25. *

✓ A. Datos Categóricos - Diagrama de Barras

```

import pandas as pd
import matplotlib.pyplot as plt

# @title **A. Datos Categóricos - Diagrama de Barras**
url = 'https://raw.githubusercontent.com/JSEFERINO/Teoria-de-Probabilidad-MEYCD/main/DATOS202460ULTIMOS.csv'
df10= pd.read_csv(url,delimiter=';')
df10

# Configuración de la figura y los subplots
fig, axes = plt.subplots(1, 3, figsize=(18, 5))

# Función para agregar etiquetas de porcentaje a las barras
def add_percentage_labels(ax, total_count):

```

```

for p in ax.patches:
    height = p.get_height()
    percentage = height / total_count * 100
    ax.annotate(f'{percentage:.1f}%',
                (p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', fontsize=10)

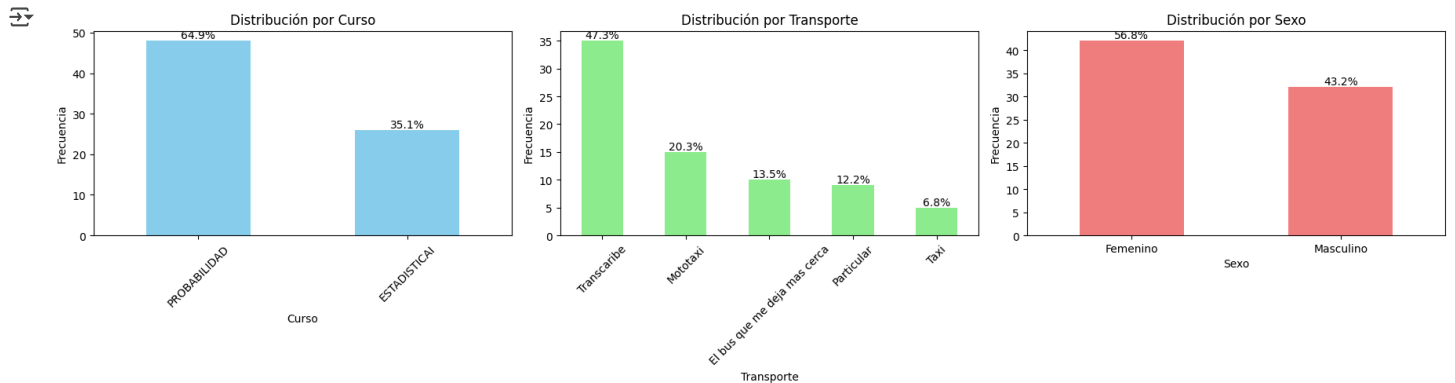
# Diagrama de barras para la variable 'CURSO'
curso_counts = df10['CURSO'].value_counts()
curso_counts.plot(kind='bar', color='skyblue', ax=axes[0])
axes[0].set_title('Distribución por Curso')
axes[0].set_xlabel('Curso')
axes[0].set_ylabel('Frecuencia')
axes[0].tick_params(axis='x', rotation=45)
add_percentage_labels(axes[0], curso_counts.sum())

# Diagrama de barras para la variable 'TRANSPORTE'
transporte_counts = df10['TRANSPORTE'].value_counts()
transporte_counts.plot(kind='bar', color='lightgreen', ax=axes[1])
axes[1].set_title('Distribución por Transporte')
axes[1].set_xlabel('Transporte')
axes[1].set_ylabel('Frecuencia')
axes[1].tick_params(axis='x', rotation=45)
add_percentage_labels(axes[1], transporte_counts.sum())

# Diagrama de barras para la variable 'SEXO'
sexo_counts = df10['SEXO'].value_counts()
sexo_counts.plot(kind='bar', color='lightcoral', ax=axes[2])
axes[2].set_title('Distribución por Sexo')
axes[2].set_xlabel('Sexo')
axes[2].set_ylabel('Frecuencia')
axes[2].tick_params(axis='x', rotation=0)
add_percentage_labels(axes[2], sexo_counts.sum())

# Ajustar el layout
plt.tight_layout()
plt.show()

```



▼ B. Creación de una tabla de contingencia para las tres variables

```

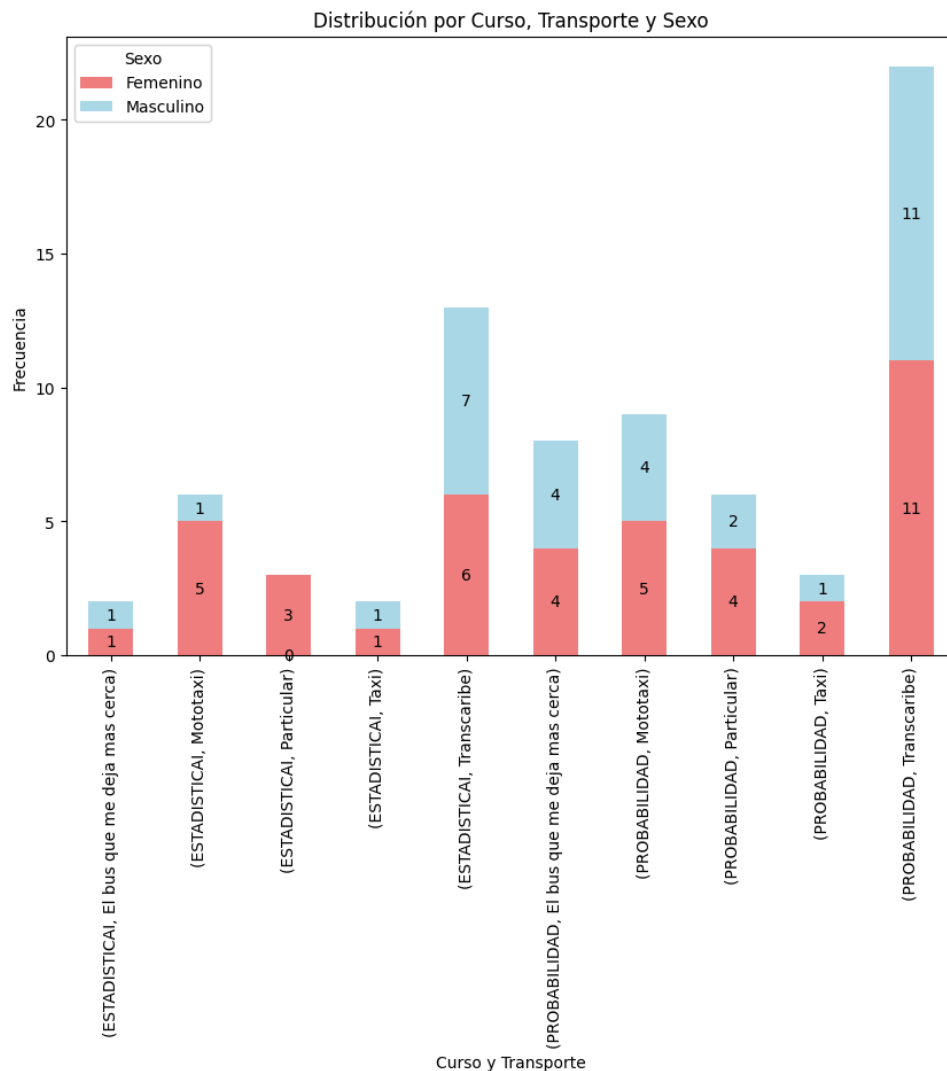
# @title **B. Creación de una tabla de contingencia para las tres variables**
contingency_table = pd.crosstab([df10['CURSO'], df10['TRANSPORTE']], df10['SEXO'])

# Configuración del gráfico
ax = contingency_table.plot(kind='bar', stacked=True, figsize=(10, 7), color=['lightcoral', 'lightblue' ])
plt.title('Distribución por Curso, Transporte y Sexo')
plt.xlabel('Curso y Transporte')
plt.ylabel('Frecuencia')
plt.xticks(rotation=90)
plt.legend(title='Sexo')

# Añadir etiquetas de datos en las barras
for p in ax.patches:
    width = p.get_width() # Anchura de la barra
    height = p.get_height() # Altura de la barra
    x = p.get_x() + width / 2 # Coordenada X del centro de la barra
    y = p.get_y() + height / 2 # Coordenada Y del centro de la barra
    ax.text(x, y, f'{int(height)}', ha='center', va='center')

plt.show()

```



✓ C. Creación de la tabla de contingencia para SEXO vs TRANSPORTE

```
# @title **C. Creación de la tabla de contingencia para SEXO vs TRANSPORTE**
contingency_table = pd.crosstab(df10['TRANSPORTE'], df10['SEXO'])

# Calcular los porcentajes para cada categoría
contingency_table_percentage = contingency_table.div(contingency_table.sum(axis=1), axis=0) * 100

# Crear subplots para mostrar cada transporte por separado
fig, axs = plt.subplots(1, len(contingency_table), figsize=(18, 10))

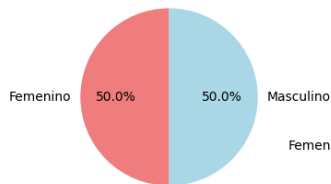
# Generar un gráfico circular por cada tipo de transporte
for i, transporte in enumerate(contingency_table.index):
    axs[i].pie(contingency_table_percentage.loc[transporte],
               labels=contingency_table.columns,
               autopct='%1.1f%%',
               startangle=90,
               colors=['lightcoral', 'lightblue'])
    axs[i].set_title(f'Transporte: {transporte}')

plt.suptitle('Distribución por Sexo en función del Transporte')
plt.show()
```

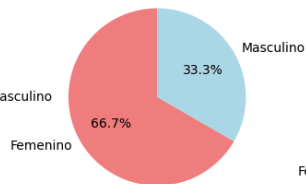


Distribución por Sexo en función del Transporte

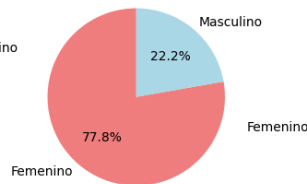
Transporte: El bus que me deja mas cerca



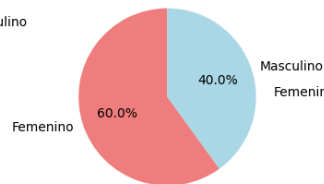
Transporte: Mototaxi



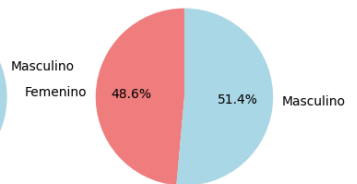
Transporte: Particular



Transporte: Taxi



Transporte: Transcribire



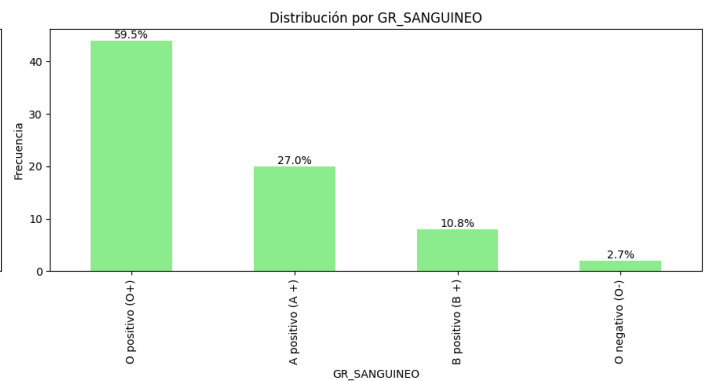
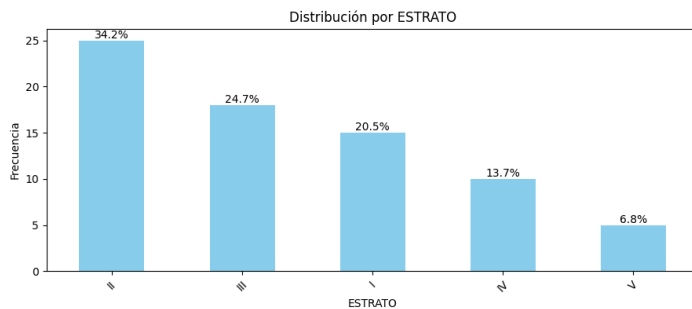
```
# Configuración de la figura y los subplots
fig, axes = plt.subplots(1, 2, figsize=(18, 5))

# Función para agregar etiquetas de porcentaje a las barras
def add_percentage_labels(ax, total_count):
    for p in ax.patches:
        height = p.get_height()
        percentage = height / total_count * 100
        ax.annotate(f'{percentage:.1f}%',
                    (p.get_x() + p.get_width() / 2., height),
                    ha='center', va='bottom', fontsize=10)

# Diagrama de barras para la variable 'CURSO'
curso_counts = df10['ESTRATO'].value_counts()
curso_counts.plot(kind='bar', color='skyblue', ax=axes[0])
axes[0].set_title('Distribución por ESTRATO')
axes[0].set_xlabel('ESTRATO')
axes[0].set_ylabel('Frecuencia')
axes[0].tick_params(axis='x', rotation=45)
add_percentage_labels(axes[0], curso_counts.sum())

# Diagrama de barras para la variable 'TRANSPORTE'
transporte_counts = df10['GR_SANGUINEO'].value_counts()
transporte_counts.plot(kind='bar', color='lightgreen', ax=axes[1])
axes[1].set_title('Distribución por GR_SANGUINEO')
axes[1].set_xlabel('GR_SANGUINEO')
axes[1].set_ylabel('Frecuencia')
axes[1].tick_params(axis='x', rotation=90)
add_percentage_labels(axes[1], transporte_counts.sum())

# Ajustar el layout
plt.tight_layout()
plt.show()
```


[Volver al inicio](#)

26. Datos agrupados para variables cuantitativa continuas

A. Datos agrupados para variables cuantitativa continuas - Variable "PESO"


```
# @title **A. Datos agrupados para variables cuantitativa continuas - Variable "PESO"**
```

```
import pandas as pd
```

```
peso = df10['PESO']
peso.head()
```

```
↗
```

	PESO
0	55
1	80
2	60
3	72
4	45

```
dtype: int64
```

▼ B. Diagrama de tallo y hoja

```
# @title **B. Diagrama de tallo y hoja**
def stem_leaf_plot(data):
    stems = {}
    for value in data:
        # Handle potential floats by converting to integer if needed or appropriate
        # Assuming data is integers or can be treated as integers for this plot
        value_int = int(value)
        stem = value_int // 10
        leaf = value_int % 10
        if stem not in stems:
            stems[stem] = [leaf]
        else:
            stems[stem].append(leaf)

    # Sort the stems and leaves before printing
    sorted_stems = sorted(stems.keys())

    for stem in sorted_stems:
        sorted_leaves = sorted(stems[stem])
        print(f"{stem} | {' '.join(map(str, sorted_leaves))}")

stem_leaf_plot(peso)
```

```
↗
```

```
4 | 2 5 8 8 8 9
5 | 0 0 0 0 0 0 2 3 3 4 5 5 5 5 6 7 7 8 9 9
6 | 0 0 0 0 0 0 0 0 1 2 2 2 3 4 4 5 5 5 5 5 7 7 7 9 9
7 | 0 0 0 0 0 1 2 2 3 5 6 7 8 8
8 | 0 0 0 2 8
9 | 0
10 | 2
```

▼ C. Pasos para construir la tabla agrupada

```
# @title **C. Pasos para construir la tabla agrupada**
# Definir el número de intervalos (se puede ajustar según la cantidad de datos)
num_intervalos = round(1+3.322*np.log10(len(peso)),0)

print(f'Numero de intervalos, c = {num_intervalos}')

# Calcular el rango de los datos
rango = max(peso) - min(peso)
print(f'Rango de los datos, R = {max(peso)}-{min(peso)}={rango}')

# Calcular el ancho del intervalo
ancho_intervalo = round(rango / num_intervalos,1)
print(f'Ancho del intervalo, h = {ancho_intervalo}')
```

```
↗
```

```
Numero de intervalos, c = 7.0
Rango de los datos, R = 102-42=60
Ancho del intervalo, h = 8.6
```

▼ D. Crear los intervalos

```
# @title **D. Crear los intervalos**

num_intervalos = round(1+3.322*np.log10(len(peso)),0)

intervalos = []
inicio_intervalo = min(peso)
for i in range(int(num_intervalos)):
    fin_intervalo = inicio_intervalo + ancho_intervalo
    # Convert the Series to scalar values using .item() before storing
```

```

intervalos.append((pd.Series(inicio_intervalo).round(1).item(), pd.Series(fin_intervalo).round(1).item()))
inicio_intervalo = fin_intervalo

# Inicializar las listas para almacenar la información de la tabla
marca_clase = []
frecuencia_absoluta = []
frecuencia_absoluta_acumulada = []
frecuencia_relativa = []
frecuencia_relativa_acumulada = []

frecuencia_acumulada = 0
for intervalo in intervalos:
    # Calcular la marca de clase - Now interval elements are scalars
    marca = (intervalo[0] + intervalo[1]) / 2
    marca_clase.append(marca)

    # Contar la frecuencia absoluta del intervalo - Comparison with scalar values from interval tuple
    frecuencia = sum(1 for peso_i in peso if intervalo[0] <= peso_i < intervalo[1])
    frecuencia_absoluta.append(frecuencia)

    # Calcular la frecuencia absoluta acumulada
    frecuencia_acumulada += frecuencia
    frecuencia_absoluta_acumulada.append(frecuencia_acumulada)

    # Calcular la frecuencia relativa
    frecuencia_relativa.append(frecuencia / len(peso))

    # Calcular la frecuencia relativa acumulada
    frecuencia_relativa_acumulada.append(frecuencia_acumulada / len(peso))

# Crear un DataFrame de Pandas con la tabla de frecuencias
# Ensure marca_clase elements are scalars before creating Series
tabla_frecuencias = pd.DataFrame({
    'Intervalo': intervalos,
    # Ensure Marca de Clase column is created from scalar values
    'Marca de Clase': [m.item() if isinstance(m, pd.Series) else m for m in marca_clase],
    'Fre_Abs': frecuencia_absoluta,
    'Fre_Abs_Acum': pd.Series(frecuencia_absoluta_acumulada).round(3),
    'Frec_Rel': pd.Series(frecuencia_relativa).round(3),
    'Fre_Rel_Acum': pd.Series(frecuencia_relativa_acumulada).round(3)
})

tabla_frecuencias

```

	Intervalo	Marca de Clase	Fre_Abs	Fre_Abs_Acum	Frec_Rel	Fre_Rel_Acum
0	(42, 50.6)	46.3	12	12	0.162	0.162
1	(50.6, 59.2)	54.9	15	27	0.203	0.365
2	(59.2, 67.8)	63.5	24	51	0.324	0.689
3	(67.8, 76.4)	72.1	13	64	0.176	0.865
4	(76.4, 85.0)	80.7	7	71	0.095	0.959
5	(85.0, 93.6)	89.3	2	73	0.027	0.986
6	(93.6, 102.2)	97.9	1	74	0.014	1.000

[Volver al inicio](#)

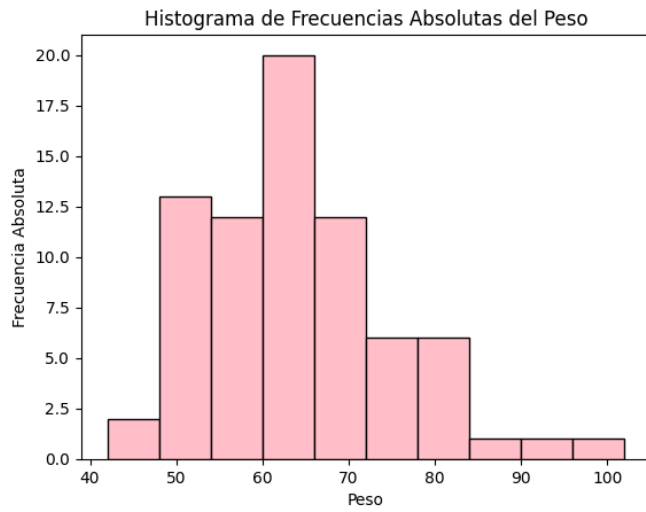
27. Histogramas y Polígonos de Frecuencia absolutas

A. Histograma de frecuencias absolutas

```

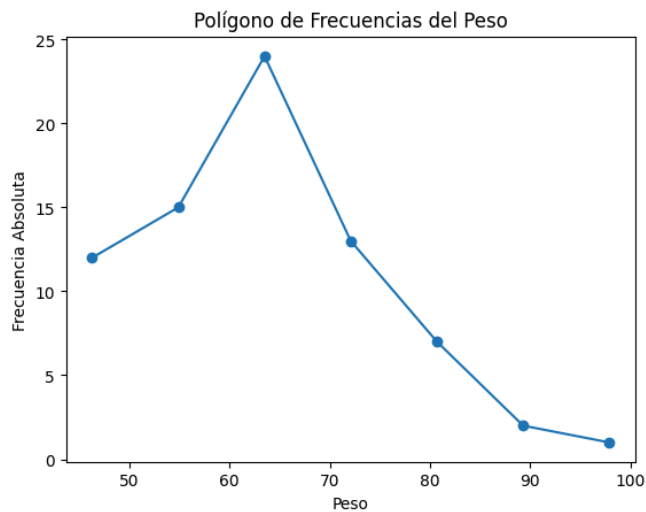
# @title **A. Histograma de frecuencias absolutas**
plt.hist(peso, bins=10, color='pink', edgecolor='black')
plt.xlabel('Peso')
plt.ylabel('Frecuencia Absoluta')
plt.title('Histograma de Frecuencias Absolutas del Peso')
plt.show()

```



▼ B. Polígono de frecuencias

```
# @title **B. Polígono de frecuencias**
plt.plot(tabla_frecuencias['Marca de Clase'], tabla_frecuencias['Fre_Abs'], marker='o')
plt.xlabel('Peso')
plt.ylabel('Frecuencia Absoluta')
plt.title('Polígono de Frecuencias del Peso')
plt.show()
```



▼ k. histograma y el polígono de frecuencias

```
# @title **k. histograma y el polígono de frecuencias**

# Integrar los gráficos del histograma y el polígono de frecuencias
plt.figure(figsize=(10, 6))

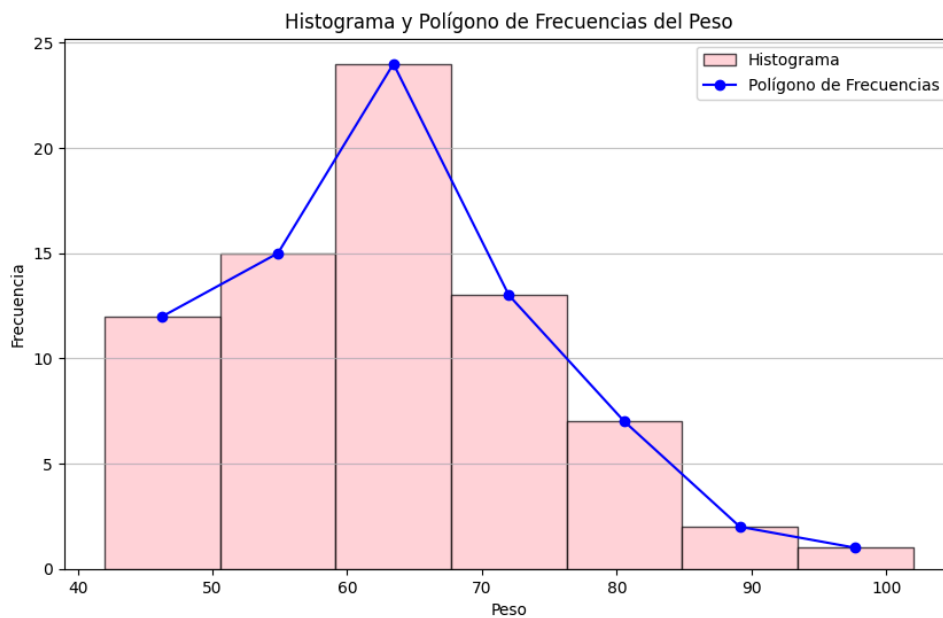
# Histograma
plt.hist(peso, bins=int(num_intervalos), color='pink', edgecolor='black', alpha=0.7, label='Histograma') # Use num_intervalos directly

# Polígono de frecuencias (usando los puntos medios de las barras)
# Calculate bin edges and then midpoints
bin_edges = np.histogram_bin_edges(peso, bins=int(num_intervalos))
bin_midpoints = (bin_edges[:-1] + bin_edges[1:]) / 2

# Calculate frequencies for plotting the polygon
hist, _ = np.histogram(peso, bins=bin_edges)

plt.plot(bin_midpoints, hist, marker='o', color='blue', linestyle='-', label='Polígono de Frecuencias')

plt.xlabel('Peso')
plt.ylabel('Frecuencia')
plt.title('Histograma y Polígono de Frecuencias del Peso')
plt.legend()
plt.grid(axis='y', alpha=0.75)
plt.show()
```



▼ C. histograma y el polígono de frecuencias

```
# @title **C. histograma y el polígono de frecuencias**

# Integrar los gráficos del histograma y el polígono de frecuencias
plt.figure(figsize=(10, 6))

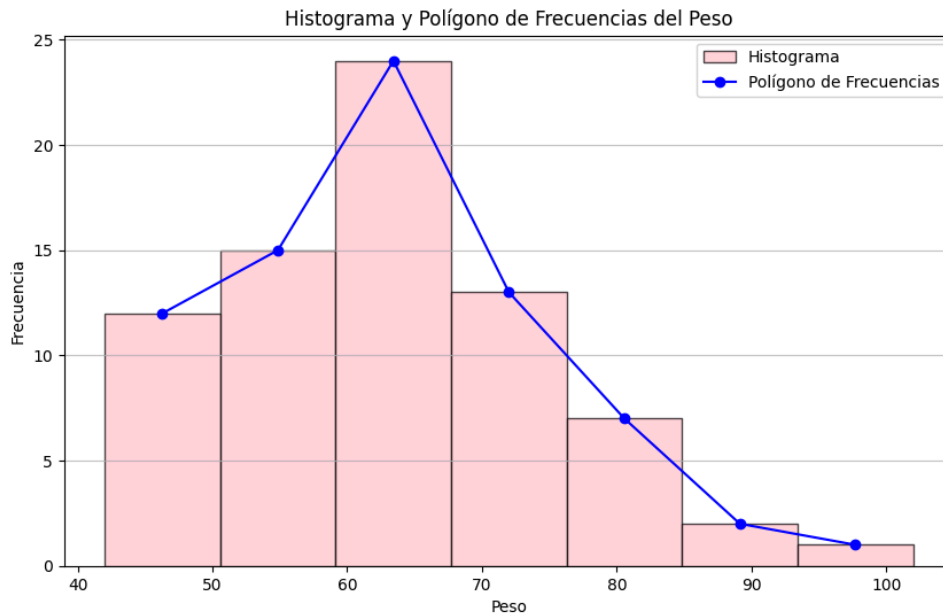
# Histograma
plt.hist(peso, bins=int(num_intervalos), color='pink', edgecolor='black', alpha=0.7, label='Histograma') # Use num_intervalos directly

# Polígono de frecuencias (usando los puntos medios de las barras)
# Calculate bin edges and then midpoints
bin_edges = np.histogram_bin_edges(peso, bins=int(num_intervalos))
bin_midpoints = (bin_edges[:-1] + bin_edges[1:]) / 2

# Calculate frequencies for plotting the polygon
hist, _ = np.histogram(peso, bins=bin_edges)

plt.plot(bin_midpoints, hist, marker='o', color='blue', linestyle='-', label='Polígono de Frecuencias')

plt.xlabel('Peso')
plt.ylabel('Frecuencia')
plt.title('Histograma y Polígono de Frecuencias del Peso')
plt.legend()
plt.grid(axis='y', alpha=0.75)
plt.show()
```



[Volver al inicio](#)

28. Histograma de frecuencias relativas

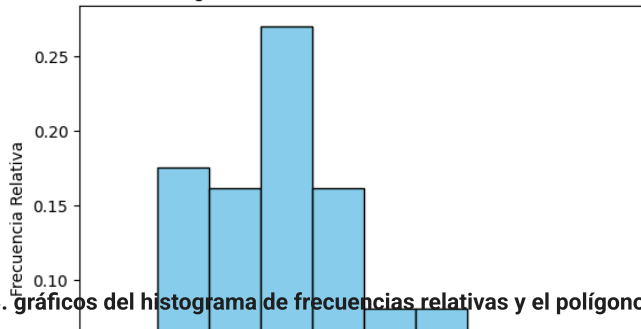
A. Histograma de frecuencias relativas

```
# @title **A. Histograma de frecuencias relativas**
plt.hist(peso, bins=10, weights=[1/len(peso)]*len(peso), color='skyblue', edgecolor='black')
plt.xlabel('Peso')
plt.ylabel('Frecuencia Relativa')
plt.title('Histograma de Frecuencias Relativas del Peso')
plt.show()

# Polígono de frecuencias relativas
plt.plot(tabla_frecuencias['Marca de Clase'], tabla_frecuencias['Frec_Rel'], marker='o')
plt.xlabel('Peso')
plt.ylabel('Frecuencia Relativa')
plt.title('Polígono de Frecuencias Relativas del Peso')
plt.show()
```



Histograma de Frecuencias Relativas del Peso



▼ B. gráficos del histograma de frecuencias relativas y el polígono de frecuencias relativas

```
# @title **B. gráficos del histograma de frecuencias relativas y el polígono de frecuencias relativas**
plt.figure(figsize=(10, 6))

# Histograma de frecuencias relativas
# Calculate bin edges and then midpoints
bin_edges = np.histogram_bin_edges(peso, bins=int(num_intervalos))
bin_midpoints = (bin_edges[:-1] + bin_edges[1:]) / 2

# Calculate frequencies for plotting the polygon
hist, _ = np.histogram(peso, bins=bin_edges)
relative_hist = hist / len(peso)

plt.hist(peso, bins=int(num_intervalos), weights=np.ones_like(peso) / len(peso), color='skyblue', edgecolor='black', alpha=0.7, label='Histograma Relativo')

# Polígono de frecuencias relativas
plt.plot(bin_midpoints, relative_hist, marker='o', color='red', linestyle='-', label='Polígono de Frecuencias Relativas')

plt.xlabel('Peso')
plt.ylabel('Frecuencia Relativa')
plt.title('Histograma y Polígono de Frecuencias Relativas del Peso')
plt.legend()
plt.grid(axis='y', alpha=0.75)
plt.show()
```



Histograma y Polígono de Frecuencias Relativas del Peso

