Limpieza y análisis descriptivo de los datos

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score
from IPython.display import HTML, display_html, display
```

```
# write here you code
df = pd.read_excel('/content/diabetes_cortesia_promitat-1.xlsx')
df.head()
```

	Unnamed: 0	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	altura	peso	IM
0	1	193	77	49	3.9	19	female	61	119	22.
1	2	146	79	41	3.6	19	female	60	135	26.
2	3	217	75	54	4.0	20	female	67	187	29.
3	4	226	97	70	3.2	20	female	64	114	19.

```
remove_cols = [col for col in df.columns if 'Unnamed' in col]
df.drop(remove_cols, axis='columns', inplace=True)
dfdibetesgen = df.copy()
df_regresion = df.copy()
df_discriminante = df.copy()
df = df.drop(['genero'], axis = 1)
df_con_diabetes = df.copy()
```

```
df
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_cin_cad	
0	193	77	49	3.9	19	61	119	22.5	118	70	32	38	0.84	I
1	146	79	41	3.6	19	60	135	26.4	108	58	33	40	0.83	I
2	217	75	54	4.0	20	67	187	29.3	110	72	40	45	0.89	I
3	226	97	70	3.2	20	64	114	19.6	122	64	31	39	0.79	I
4	164	91	67	2.4	20	70	141	20.2	122	86	32	39	0.82	I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
385	227	105	44	5.2	83	59	125	25.2	150	90	35	40	0.88	I
386	226	279	52	4.3	84	60	192	37.5	144	88	41	48	0.85	
387	301	90	118	2.6	89	61	115	21.7	218	90	31	41	0.76	I
388	232	184	114	2.0	91	61	127	24.0	170	82	35	38	0.92	
389	165	94	69	2.4	92	62	217	39.7	160	82	51	51	1.00	I

390 rows x 14 columns

```
df.corr()
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura
colesterol	1.000000	0.158102	0.193162	0.475927	0.247333	-0.063601	0.062359	0.091695	0.207741	0.166241	0.134038
glucosa	0.158102	1.000000	-0.158302	0.282210	0.294392	0.098052	0.190358	0.129286	0.162777	0.020262	0.222336
hdl_chol	0.193162	-0.158302	1.000000	-0.681867	0.028210	-0.087238	-0.291883	-0.241860	0.031807	0.078342	-0.276697
prop_col_hdl	0.475927	0.282210	-0.681867	1.000000	0.163201	0.081162	0.278812	0.228407	0.115505	0.038242	0.313262
edad	0.247333	0.294392	0.028210	0.163201	1.000000	-0.082229	-0.056784	-0.009164	0.453417	0.068649	0.150585
altura	-0.063601	0.098052	-0.087238	0.081162	-0.082229	1.000000	0.255389	-0.259589	-0.040704	0.043617	0.057447
peso	0.062359	0.190358	-0.291883	0.278812	-0.056784	0.255389	1.000000	0.860147	0.097497	0.166477	0.847766
IMC	0.091695	0.129286	-0.241860	0.228407	-0.009164	-0.259589	0.860147	1.000000	0.121408	0.145304	0.810701
ps_sistolica	0.207741	0.162777	0.031807	0.115505	0.453417	-0.040704	0.097497	0.121408	1.000000	0.603662	0.210934
ps_diastolica	0.166241	0.020262	0.078342	0.038242	0.068649	0.043617	0.166477	0.145304	0.603662	1.000000	0.165846
cintura	0.134038	0.222336	-0.276697	0.313262	0.150585	0.057447	0.847766	0.810701	0.210934	0.165846	1.000000
caderas	0.093364	0.138223	-0.223837	0.208902	0.004675	-0.095906	0.826985	0.881728	0.155321	0.143898	0.835177
prop_cin_cad	0.091847	0.185117	-0.158777	0.243329	0.275188	0.252548	0.250461	0.100873	0.137871	0.077918	0.514177

```
df = df.drop(['diabetes'], axis = 1)
df.head()
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_cin_cad
0	193	77	49	3.9	19	61	119	22.5	118	70	32	38	0.84
1	146	79	41	3.6	19	60	135	26.4	108	58	33	40	0.83
2	217	75	54	4.0	20	67	187	29.3	110	72	40	45	0.89
3	226	97	70	3.2	20	64	114	19.6	122	64	31	39	0.79
4	164	91	67	2.4	20	70	141	20.2	122	86	32	39	0.82

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 390 entries, 0 to 389
Data columns (total 13 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   colesterol     390 non-null    int64
 1   glucosa        390 non-null    int64
 2   hdl_chol       390 non-null    int64
 3   prop_col_hdl   390 non-null    float64
 4   edad           390 non-null    int64
 5   altura         390 non-null    int64
 6   peso           390 non-null    int64
 7   IMC            390 non-null    float64
 8   ps_sistolica   390 non-null    int64
 9   ps_diastolica  390 non-null    int64
 10  cintura        390 non-null    int64
 11  caderas        390 non-null    int64
 12  prop_cin_cad   390 non-null    float64
dtypes: float64(3), int64(10)
memory usage: 39.7 KB
```

```
df.isna().sum()
```

```
colesterol       0
glucosa          0
hdl_chol         0
prop_col_hdl     0
edad             0
altura           0
peso             0
IMC              0
ps_sistolica     0
ps_diastolica    0
cintura          0
caderas          0
```

```
prop_cin_cad     0
dtype: int64
```

```
df.shape
```

```
(390, 13)
```

```
df.describe()
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	ci
count	390.000000	390.000000	390.000000	390.000000	390.000000	390.000000	390.000000	390.000000	390.000000	390.000000	390.0
mean	207.230769	107.338462	50.266667	4.524615	46.774359	65.951282	177.407692	28.775641	137.133333	83.289744	37.8
std	44.666005	53.798188	17.279069	1.736634	16.435911	3.918867	40.407824	6.600915	22.859528	13.498192	5.7
min	78.000000	48.000000	12.000000	1.500000	19.000000	52.000000	99.000000	15.200000	90.000000	48.000000	26.0
25%	179.000000	81.000000	38.000000	3.200000	34.000000	63.000000	150.250000	24.100000	122.000000	75.000000	33.0
50%	203.000000	90.000000	46.000000	4.200000	44.500000	66.000000	173.000000	27.800000	136.000000	82.000000	37.0
75%	229.000000	107.750000	59.000000	5.400000	60.000000	69.000000	200.000000	32.275000	148.000000	90.000000	41.0
max	443.000000	385.000000	120.000000	19.300000	92.000000	76.000000	325.000000	55.800000	250.000000	124.000000	56.0

```
varianza = df.var()
display(HTML('<h1 style = "color:#FF85C0";> <FONT FACE="optima">Varianzas</FONT></h1>'))
display_html(pd.DataFrame(varianza).style.set_table_styles([{'selector': 'th:not(.index_name)', 'props': 'background-color: #FF85C0; color
```

Varianzas

	0
colesterol	1995.052007
glucosa	2894.245046
hdl_chol	298.566238
prop_col_hdl	3.015896
edad	270.139187
altura	15.357518
peso	1632.792229
IMC	43.572079
ps_sistolica	522.558012
ps_diastolica	182.201180
cintura	33.188511
caderas	32.084774
prop_cin_cad	0.005360

```
cv = df.std() / df.mean() * 100
display(HTML('<h1 style = "color:#C0FF85";> <FONT FACE="optima">Coeficientes de variación</FONT></h1>'))
display_html(pd.DataFrame(cv).style.set_table_styles([{'selector': 'th:not(.index_name)', 'props': 'background-color: #C0FF85; color: whit
```

Coeficientes de variación

	0
colesterol	21.553752
glucosa	50.120141
hdl_chol	34.374806
prop_col_hdl	38.381906
edad	35.138721
altura	5.942063

```
# pd.DataFrame(np.cov(df.T, bias = True)).style.set_table_styles([{'selector': 'th:not(.index_name)', 'props': 'background-color: #FF85FD;
display(HTML('<h1 style = "color:#FF85FD";> <FONT FACE="optima">Matriz de varianzas y covarianzas S</FONT></h1>'))
display_html(pd.DataFrame(np.cov(df.T, bias = True)).style.set_table_styles([{'selector': 'th:not(.index_name)', 'props': 'background-colo
```

Matriz de varianzas y covarianzas S

	0	1	2	3	4	5	6	7	8	9	10
0	1989.936489	378.937278	148.697436	36.822268	181.108481	-11.104142	112.259763	26.965621	211.569231	99.971598	34.401972
1	378.937278	2886.823905	-146.777436	26.298592	259.640473	20.619053	412.751755	45.794142	199.670256	14.676292	68.731440
2	148.697436	-146.777436	297.800684	-20.408615	7.990940	-5.892137	-203.272821	-27.515299	12.531111	18.225299	-27.472821
3	36.822268	26.298592	-20.408615	3.008163	4.646323	0.550943	19.515093	2.611600	4.573641	0.894150	3.126039
4	181.108481	259.640473	7.990940	4.646323	269.446522	-5.282788	-37.615700	-0.991650	169.919829	15.191019	14.221775
5	-11.104142	20.619053	-5.892137	0.550943	-5.282788	15.318139	40.337811	-6.697853	-3.637094	2.301295	1.293629
6	112.259763	412.751755	-203.272821	19.515093	-37.615700	40.337811	1628.605582	228.837623	89.827692	90.569053	196.843057
7	26.965621	45.794142	-27.515299	2.611600	-0.991650	-6.697853	228.837623	43.460355	18.272735	12.913468	30.749892
8	211.569231	199.670256	12.531111	4.573641	169.919829	-3.637094	89.827692	18.272735	521.218120	185.789573	27.707179
9	99.971598	14.676292	18.225299	0.894150	15.191019	2.301295	90.569053	12.913468	185.789573	181.733997	12.863531
10	34.401972	68.731440	-27.472821	3.126039	14.221775	1.293629	196.843057	30.749892	27.707179	12.863531	33.103412
11	23.560750	42.012860	-21.851795	2.049677	0.434162	-2.123452	188.798008	32.883146	20.060000	10.974024	27.183609
12	0.299578	0.727249	-0.200344	0.030858	0.330287	0.072273	0.739051	0.048623	0.230149	0.076804	0.216309

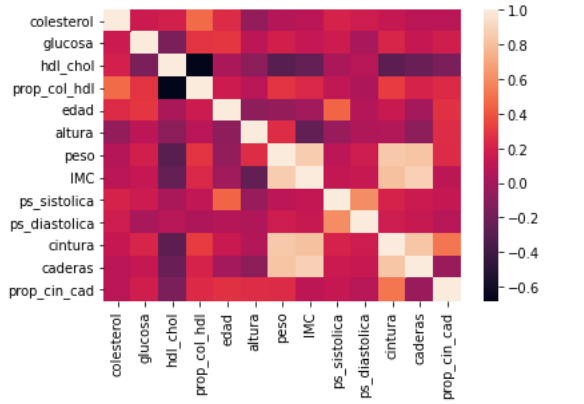
```
# Matriz de correlación R
display(HTML('<h1 style = "color:#FFC485";> <FONT FACE="optima">Matriz de correlación R</FONT></h1>'))
display_html(df.corr(method = 'pearson').style.set_table_styles([{'selector': 'th:not(.index_name)', 'props': 'background-color: #FFC485;
```

Matriz de correlación R

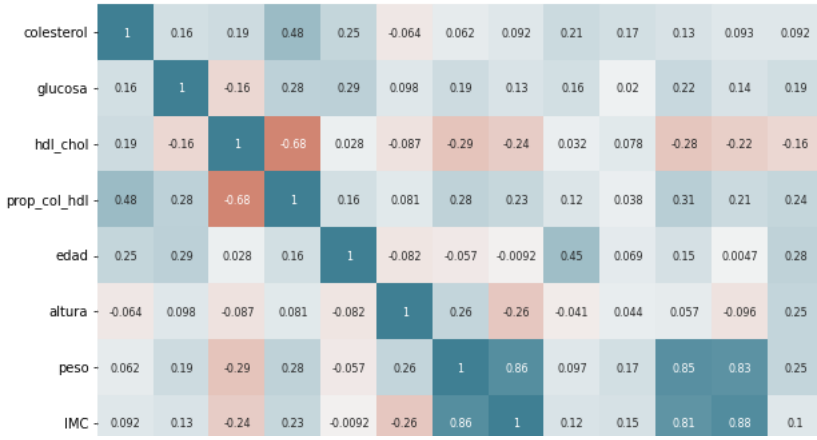
	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diast
colesterol	1.000000	0.158102	0.193162	0.475927	0.247333	-0.063601	0.062359	0.091695	0.207741	0.
glucosa	0.158102	1.000000	-0.158302	0.282210	0.294392	0.098052	0.190358	0.129286	0.162777	0.
hdl_chol	0.193162	-0.158302	1.000000	-0.681867	0.028210	-0.087238	-0.291883	-0.241860	0.031807	0.
prop_col_hdl	0.475927	0.282210	-0.681867	1.000000	0.163201	0.081162	0.278812	0.228407	0.115505	0.

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.heatmap(df_regresion.corr())
```

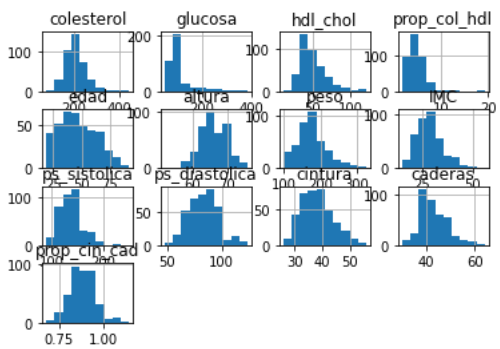
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f659e27f090>
```




```
# Heatmap matriz de correlaciones
# ==============================================================================
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(12, 10))
corr_matrix = df_regresion.corr(method='pearson')
sns.heatmap(
    corr_matrix,
    annot     = True,
    cbar      = False,
    annot_kws = {"size": 8},
    vmin      = -1,
    vmax      = 1,
    center    = 0,
    cmap      = sns.diverging_palette(20, 220, n=200),
    square    = True,
    ax        = ax
)

ax.set_xticklabels(
    ax.get_xticklabels(),
    rotation = 45,
    horizontalalignment = 'right',
)

ax.tick_params(labelsize = 10)
```




```
df_regresion.drop(['diabetes'], axis = 1).hist()
plt.show()
```



Estandarización de los datos

```
from sklearn import preprocessing
z_scaler = preprocessing.StandardScaler()
df_stan = pd.DataFrame(z_scaler.fit_transform(df))
df_stan = df_stan.set_axis(['colesterol', 'glucosa', 'hdl_chol', 'prop_col_hdl', 'edad', 'altura','peso','IMC','ps_sistolica','ps_diastoli

df_stan.head()
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas
0	-0.319013	-0.564655	-0.073401	-0.360132	-1.692029	-1.265070	-1.447312	-0.951944	-0.838071	-0.985822	-1.020105	-0.882489
1	-1.372619	-0.527432	-0.536983	-0.533102	-1.692029	-1.520574	-1.050840	-0.360358	-1.276087	-1.875972	-0.846299	-0.528950
2	0.218998	-0.601879	0.216339	-0.302476	-1.631108	0.267951	0.237692	0.079539	-1.188484	-0.837464	0.370339	0.354899
3	0.420753	-0.192418	1.143504	-0.763729	-1.631108	-0.498560	-1.571209	-1.391841	-0.662865	-1.430897	-1.193910	-0.705719
4	-0.969111	-0.304089	0.969660	-1.224982	-1.631108	1.034462	-0.902163	-1.300828	-0.662865	0.201045	-1.020105	-0.705719

Componentes principales

Observar y quitar outliers

```
from scipy.stats import chi2
from matplotlib import patches
import matplotlib.pyplot as plt
# Import dataset and clean it
df1 = df_stan.to_numpy()
```

```
# Covariance matrix
covariance  = np.cov(df1 , rowvar=False)

# Covariance matrix power of -1
covariance_pm1 = np.linalg.matrix_power(covariance, -1)

# Center point
centerpoint = np.mean(df1 , axis=0)
```

```
# Distances between center point and
distances = []
for i, val in enumerate(df1):
      p1 = val
      p2 = centerpoint
      distance = (p1-p2).T.dot(covariance_pm1).dot(p1-p2)
      distances.append(distance)
distances = np.array(distances)

# Cutoff (threshold) value from Chi-Sqaure Distribution for detecting outliers
cutoff = chi2.ppf(0.95, df1.shape[1])

# Index of outliers
outlierIndexes = np.where(distances > cutoff )

print('--- Index of Outliers ----')
print(outlierIndexes)

print('--- Observations found as outlier -----')
print(df1[ distances > cutoff , :])
```

```
--- Index of Outliers ----
(array([ 10,  82,  85,  89, 116, 139, 144, 153, 162, 167, 181, 184, 231,
       233, 234, 239, 242, 256, 261, 272, 276, 281, 296, 303, 306, 308,
       310, 316, 317, 341, 346, 347, 353, 354, 365, 371, 374, 386, 387,
       388, 389]),)
--- Observations found as outlier -----
[[-7.44939129e-01 -4.35231264e-02  3.86705165e+00 -1.74389146e+00
  -1.63110827e+00  1.03446193e+00  2.37692316e-01 -2.99682511e-01
  -2.24848354e-01  2.01044544e-01 -1.51077124e-01 -3.52179786e-01
   2.54592637e-01]
 [-3.63847584e-01  8.87070037e-01  4.48129862e-01 -7.06072027e-01
  -9.60982389e-01 -1.00956671e+00  1.47666589e+00  2.20318181e+00
   1.25564665e-01  2.75223703e-01  2.62981154e+00  2.29936687e+00
   9.38415837e-01]
 [-7.00104829e-01 -3.22701076e-01 -9.42617985e-01  3.89404045e-01
  -9.00061854e-01 -7.54063133e-01  1.84835797e+00  2.40037718e+00
  -1.62650043e+00 -8.37463695e-01  1.23936721e+00  2.65290642e+00
  -1.38658304e+00]
 [-1.26053357e+00 -4.15760392e-01 -1.31348408e-01 -8.21385297e-01
  -8.39141319e-01  7.78958354e-01  3.23600837e+00  2.53689705e+00
  -1.18848416e+00  4.97761183e-01  2.45600600e+00  2.65290642e+00
   2.54592637e-01]
 [ 3.13322777e+00  1.66876829e+00 -4.79035370e-01  2.17675974e+00
  -6.56379713e-01  1.03446193e+00  2.46784476e+00  1.65710232e+00
   1.25564665e-01  2.01044544e-01  2.28220046e+00  1.06197843e+00
   2.16929760e+00]
 [-9.48417872e-02  3.56717835e+00 -4.21087543e-01  1.01120868e-01
  -5.34538643e-01  7.78958354e-01  2.74041894e+00  2.08183081e+00
  -4.96418445e-02 -2.14929361e-02  1.76078383e+00  2.12259709e+00
  -1.55701282e-01]
 [-1.23811642e+00 -8.07468530e-02 -1.05851364e+00  1.58777503e-01
  -4.12697573e-01 -3.56460253e+00  2.37692316e-01  3.00713217e+00
   4.75977685e-01 -9.56720959e-02  2.27284170e-02  1.06197843e+00
  -1.38658304e+00]
 [-6.10436230e-01 -4.34372255e-01  1.08555596e+00 -1.10966847e+00
  -4.12697573e-01  5.23454773e-01  2.14571163e+00  1.71777782e+00
   2.13167920e-01  1.09119446e+00  8.91756124e-01  1.94582731e+00
  -1.11305376e+00]
 [-6.32853380e-01 -5.08819708e-01  2.41835598e+00 -1.51326492e+00
  -3.51777038e-01  1.54546910e+00 -1.47209122e+00 -1.93792098e+00
   3.00771175e-01  2.12970270e+00 -1.71532700e+00 -1.23602867e+00
  -1.38658304e+00]
 [-8.34607728e-01 -2.49112632e-02 -4.79035370e-01 -3.02475579e-01
  -3.51777038e-01 -1.26507030e+00 -1.67032699e+00 -1.20981499e+00
  -1.49509555e+00 -1.43089697e+00 -1.54152146e+00 -2.29664733e+00
   1.21194512e+00]
 [ 2.57279903e+00 -3.78536665e-01  2.41835598e+00 -5.90758756e-01
  -2.29935968e-01 -2.54258820e+00 -1.42253228e+00 -2.84513637e-01
  -7.50467884e-01  1.09119446e+00 -1.02010483e+00 -3.52179786e-01
  -1.38658304e+00]
```

```
[-3.41430434e-01  3.09243267e-02 -3.63139716e-01 -7.18490379e-02
 -2.29935968e-01 -4.98559552e-01  3.65725939e+00  4.09929115e+00
  1.69366293e-01 -3.18209576e-01  2.62981154e+00  3.35998553e+00
 -4.29230562e-01]
[-7.44939129e-01  1.22208358e+00 -9.42617985e-01  3.31747410e-01
  1.96507777e-01  1.03446193e+00  2.12093215e+00  1.35372483e+00
  9.57795588e-01  1.16537362e+00  2.28220046e+00  3.71352508e+00
 -1.11305376e+00]
```

```
df_stan.drop(outlierIndexes[0], inplace = True,)
df.drop(outlierIndexes[0], inplace = True)
dfdibetesgen.drop(outlierIndexes[0], inplace = True)
```

```
df=df.reset_index(drop = True)
dfdibetesgen=dfdibetesgen.reset_index(drop = True)
dfdibetesgen
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_c
0	193	77	49	3.9	19	female	61	119	22.5	118	70	32	38	
1	146	79	41	3.6	19	female	60	135	26.4	108	58	33	40	
2	217	75	54	4.0	20	female	67	187	29.3	110	72	40	45	
3	226	97	70	3.2	20	female	64	114	19.6	122	64	31	39	
4	164	91	67	2.4	20	female	70	141	20.2	122	86	32	39	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
344	252	161	87	2.9	80	female	62	162	29.6	160	100	44	41	
345	271	121	40	6.8	81	female	64	158	27.1	146	76	36	43	
346	240	88	49	4.9	82	female	63	170	30.1	180	86	41	46	
347	255	112	34	7.5	82	male	66	163	26.3	179	89	37	43	
348	227	105	44	5.2	83	female	59	125	25.2	150	90	35	40	

349 rows × 15 columns

```
df_stan=df_stan.reset_index(drop = True)
```

Cálculo de componentes principales

```
matriz_covarianza = np.cov(df_stan.T, bias = True)
eigenValues, eigenVectors = np.linalg.eig(matriz_covarianza)

idx = eigenValues.argsort()[::-1]
eigenValues = eigenValues[idx]
eigenVectors = eigenVectors[:,idx]

display(HTML('<h1 style = "color:#C0FF85";> <FONT FACE="optima">Valores propios</FONT></h1>'))
print(eigenValues)
display(HTML('<h1 style = "color:#C0FF85";> <FONT FACE="optima">Vectores propios</FONT></h1>'))
display_html(pd.DataFrame(eigenVectors, columns = ['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8','PC9','PC10','PC11', 'PC12', 'PC
```

Valores propios

```
[3.39486988e+00 1.69165264e+00 1.34014395e+00 1.20211751e+00
 8.50945336e-01 7.43667242e-01 6.32972210e-01 4.76211366e-01
 2.37574308e-01 1.24356817e-01 2.86126749e-02 3.93616649e-03
 1.31812987e-03]
```

Vectores propios

	PC1	PC2	PC3	PC4	PC5	PC6	PC7	PC8	PC9	PC10	PC11	PC12
colesterol	-0.130340	-0.254835	0.044639	0.338896	0.219518	-0.744455	0.146938	0.008010	0.089183	0.032947	0.410031	0.001531
glucosa	-0.159363	-0.184488	-0.142552	0.148049	0.033842	0.096473	-0.452438	0.825956	0.000524	0.001424	0.030630	0.005572
hdl_chol	0.201318	-0.083406	0.386347	-0.151394	0.601240	-0.212420	-0.050836	0.108626	0.021570	-0.051396	-0.596720	-0.009696
prop_col_hdl	-0.248051	-0.101966	-0.339894	0.315274	-0.390304	-0.268307	0.135266	-0.038944	-0.024345	-0.038047	-0.683277	-0.003274
edad	-0.141802	-0.507275	-0.103662	0.281801	0.286366	0.289543	-0.325167	-0.432673	-0.409883	-0.073358	0.010435	-0.009655
altura	-0.030649	-0.119991	-0.366624	-0.646766	-0.049466	-0.360297	-0.405049	-0.160279	-0.005151	0.042487	0.006865	0.319200

```
eigenVectors = pd.DataFrame(eigenVectors)
eigenVectors2 = eigenVectors.drop([12, 11, 10], axis = 1)
```

IMC	-0.400785	0.277453	0.185164	0.102431	0.091468	0.053623	0.048444	0.010608	-0.052137	-0.498423	0.019114	0.651833

```
a = int(np.sum(eigenValues))
b = eigenVectors2.shape[1]
a == b
```

```
True
```

```
eigenVectors = eigenVectors.to_numpy()
np.linalg.norm(eigenVectors[:,0])
```

```
1.0
```

```
componentes_principales = eigenValues[0:7]
componentes_principales
```

```
array([3.39486988, 1.69165264, 1.34014395, 1.20211751, 0.85094534,
       0.74366724, 0.63297221])
```

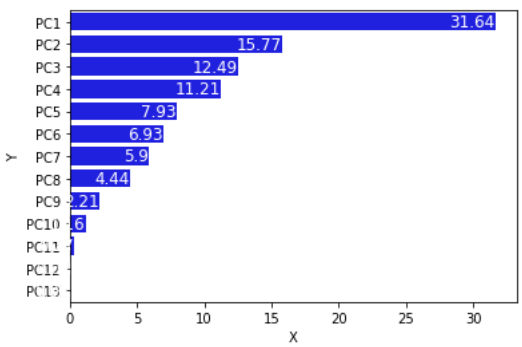
```
print('\033[1m','Varianza de los primeros tres: ','\033[0m',"{:.2f}".format(sum(eigenValues[0:7]) / sum(eigenValues) * 100), '%')
```

```
 Varianza de los primeros tres:   91.87 %
```

```
print('\033[1m','Primer eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[0] / sum(eigenValues)) * 100), '%')
print('\033[1m','Segundo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[1] / sum(eigenValues)) * 100), '%')
print('\033[1m','Tercero eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[2] / sum(eigenValues)) * 100), '%')
print('\033[1m','Cuarto eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[3] / sum(eigenValues)) * 100), '%')
print('\033[1m','Quinto eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[4] / sum(eigenValues)) * 100), '%')
print('\033[1m','Sexto eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[5] / sum(eigenValues)) * 100), '%')
print('\033[1m','Séptimo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[6] / sum(eigenValues)) * 100), '%')
print('\033[1m','Octavo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[7] / sum(eigenValues)) * 100), '%')
print('\033[1m','Noveno eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[8] / sum(eigenValues)) * 100), '%')
print('\033[1m','Décimo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[9] / sum(eigenValues)) * 100), '%')
print('\033[1m','Onceavo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[10] / sum(eigenValues)) * 100), '%')
print('\033[1m','Doceavo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[11] / sum(eigenValues)) * 100), '%')
print('\033[1m','Treceavo eigenvalor: ','\033[0m',"{:.2f}".format((eigenValues[12] / sum(eigenValues)) * 100), '%')
```

```
 Primer eigenvalor:   31.64 %
 Segundo eigenvalor:   15.77 %
 Tercero eigenvalor:   12.49 %
 Cuarto eigenvalor:   11.21 %
 Quinto eigenvalor:   7.93 %
 Sexto eigenvalor:   6.93 %
 Séptimo eigenvalor:   5.90 %
 Octavo eigenvalor:   4.44 %
 Noveno eigenvalor:   2.21 %
 Décimo eigenvalor:   1.16 %
 Onceavo eigenvalor:   0.27 %
 Doceavo eigenvalor:   0.04 %
 Treceavo eigenvalor:   0.01 %
```

```
labels=[]
for i in range(len(eigenValues)):
  labels.append(round((eigenValues[i] / sum(eigenValues)) * 100,2))
import seaborn as sns
valgraf=[]
for i in range(len(eigenValues)):
  valgraf.append(eigenValues[i] / sum(eigenValues) * 100)
valgraf = pd.DataFrame(data    = valgraf)
valgraf.insert(1,"ss",['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8','PC9','PC10','PC11','PC12', 'PC13'],allow_duplicates=False)
valgraf=valgraf.rename({0:"X","ss":"Y"}, axis='columns')
sns.barplot(x = 'X', y = 'Y', data = valgraf, color = 'blue')

ax = plt.gca()
rects = ax.patches
for p,label in zip(ax.patches,labels):
    ax.text(p.get_width(), p.get_y() + p.get_height()/2., label,
            fontsize=12, color='white', ha='right', va='center')

```




```
print('\033[1m','Varianza acumulativa explicada: ','\033[0m',"{:.2f}".format(sum(eigenValues[0:12]) / sum(eigenValues) * 100), '%')
```

Varianza acumulativa explicada: 99.99 %

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

X = df_stan
# Define the PCA object
pca = PCA()

T = pca.fit_transform(X)
```

```
df4 = pd.DataFrame(T[:, 0])
df4['T1']=T[:, 1]
df4.shape
```

```
(349, 2)
```

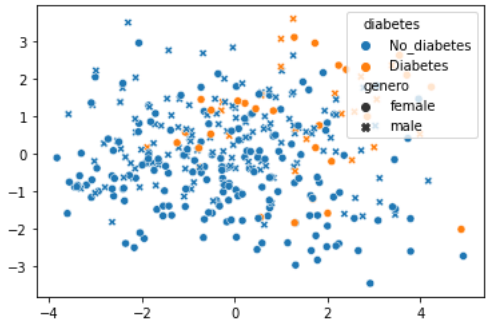
```
df_grafico = dfdibetesgen.copy()
dfdibetesgen=pd.concat([dfdibetesgen,df4],axis=1)
dfdibetesgen
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_c
0	193	77	49	3.9	19	female	61	119	22.5	118	70	32	38	
1	146	79	41	3.6	19	female	60	135	26.4	108	58	33	40	
2	217	75	54	4.0	20	female	67	187	29.3	110	72	40	45	
3	226	97	70	3.2	20	female	64	114	19.6	122	64	31	39	
4	164	91	67	2.4	20	female	70	141	20.2	122	86	32	39	

Score plot

```
sns.scatterplot(data=dfdibetesgen, x=T[:, 0], y=T[:, 1],style="genero",hue="diabetes",sizes=20)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6599406390>
```



Loading plot

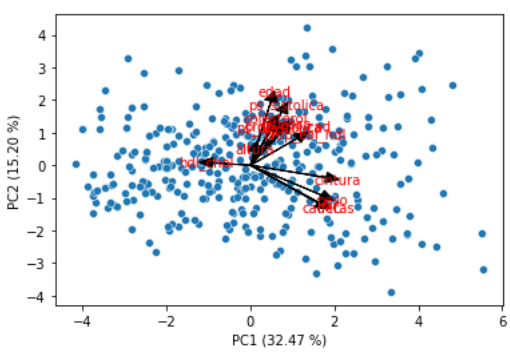
```
df_grafico = df_grafico.drop(['genero'], axis = 1)
df_grafico
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_cin_cad	
0	193	77	49	3.9	19	61	119	22.5	118	70	32	38	0.84	I
1	146	79	41	3.6	19	60	135	26.4	108	58	33	40	0.83	I
2	217	75	54	4.0	20	67	187	29.3	110	72	40	45	0.89	I
3	226	97	70	3.2	20	64	114	19.6	122	64	31	39	0.79	I
4	164	91	67	2.4	20	70	141	20.2	122	86	32	39	0.82	I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
344	252	161	87	2.9	80	62	162	29.6	160	100	44	41	1.07	
345	271	121	40	6.8	81	64	158	27.1	146	76	36	43	0.84	I
346	240	88	49	4.9	82	63	170	30.1	180	86	41	46	0.89	I
347	255	112	34	7.5	82	66	163	26.3	179	89	37	43	0.86	I
348	227	105	44	5.2	83	59	125	25.2	150	90	35	40	0.88	I

349 rows × 14 columns

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

pipeline = Pipeline([("scaler", StandardScaler()), ("pca", PCA(n_components=2)),])

pca_data = pd.DataFrame(
    pipeline.fit_transform(df_grafico.drop(columns=["diabetes"])),
    columns=["PC1", "PC2"],
    index=df_grafico.index,
)
pca_data["diabetes"] = df_grafico["diabetes"]
```

```python

pca_step = pipeline.steps[1][1]
loadings = pd.DataFrame(
    pca_step.components_.T,
    columns=["PC1", "PC2"],
    index=df_grafico.drop(columns=["diabetes"]).columns,
)


def loading_plot(
    coeff, labels, scale=1, colors=None, visible=None, ax=plt, arrow_size=0.5
):
    for i, label in enumerate(labels):
        if visible is None or visible[i]:
            ax.arrow(
                0,
                0,
                coeff[i, 0] * scale,
                coeff[i, 1] * scale,
                head_width=arrow_size * scale,
                head_length=arrow_size * scale,
                color="#000" if colors is None else colors[i],
            )
            ax.text(
                coeff[i, 0] * 1.15 * scale,
                coeff[i, 1] * 1.15 * scale,
                label,
                color="#FB0000" if colors is None else colors[i],
                ha="center",
                va="center",
            )


g = sns.scatterplot(data= pca_data, x= "PC1", y="PC2")

# Add loadings
loading_plot(loadings[["PC1", "PC2"]].values, loadings.index, scale=4, arrow_size=0.08)


# Add variance explained by the
g.set_xlabel(f"PC1 ({pca_step.explained_variance_ratio_[0]*100:.2f} %)")
g.set_ylabel(f"PC2 ({pca_step.explained_variance_ratio_[1]*100:.2f} %)")

plt.savefig("PCA_with_loadings.png", dpi=200)
plt.rcParams["figure.figsize"] = (12, 10)
plt.show()
```



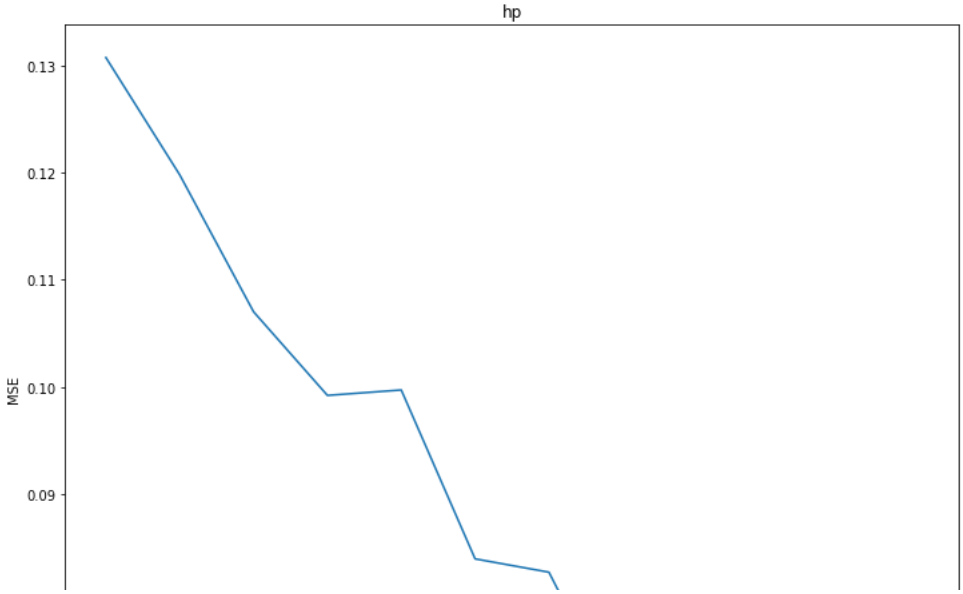
PCR (Principal Components Regression)

```python
df_con_diabetes
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_cin_cad	
0	193	77	49	3.9	19	61	119	22.5	118	70	32	38	0.84	I
1	146	79	41	3.6	19	60	135	26.4	108	58	33	40	0.83	I
2	217	75	54	4.0	20	67	187	29.3	110	72	40	45	0.89	I
3	226	97	70	3.2	20	64	114	19.6	122	64	31	39	0.79	I
4	164	91	67	2.4	20	70	141	20.2	122	86	32	39	0.82	I
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
385	227	105	44	5.2	83	59	125	25.2	150	90	35	40	0.88	I
386	226	279	52	4.3	84	60	192	37.5	144	88	41	48	0.85	
387	301	90	118	2.6	89	61	115	21.7	218	90	31	41	0.76	I

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale
from sklearn import model_selection
from sklearn.model_selection import RepeatedKFold
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

#select subset of data
col = "diabetes"
for i in range(df_con_diabetes.shape[0]):
  if df_con_diabetes[col][i] == "No_diabetes":
    df_con_diabetes[col][i] = 0
  elif df_con_diabetes[col][i] == "Diabetes":
    df_con_diabetes[col][i] = 1

data = df_con_diabetes

#define predictor and response variables
X = data.iloc[:,0:12]
Y = data.iloc[:,13]

#scale predictor variables
pca = PCA()
X_reduced = pca.fit_transform(scale(X))

#define cross validation method
cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=42)

regr = LinearRegression()
mse = []

# Calculate MSE with only the intercept
score = -1*model_selection.cross_val_score(regr, np.ones((len(X_reduced),1)), Y, cv=cv, scoring='neg_mean_squared_error').mean()
mse.append(score)

# Calculate MSE using cross-validation, adding one component at a time
for i in np.arange(1, 12):
    score = -1*model_selection.cross_val_score(regr, X_reduced[:,:i], Y, cv=cv, scoring='neg_mean_squared_error').mean()
    mse.append(score)

# Plot cross-validation results
plt.plot(mse)
plt.xlabel('Number of Principal Components')
plt.ylabel('MSE')
plt.title('hp')
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
  app.launch_new_instance()
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-co
Text(0.5, 1.0, 'hp')
```




```
#calculate percentage of variation explained
np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

```
array([ 33.15,  49.52,  63.03,  73.03,  81.29,  89.34,  94.76,  97.06,
        98.31,  99.4 ,  99.96, 100.01])
```

```
#split the dataset into training (70%) and testing (30%) sets
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.3,random_state=0)

#scale the training and testing data
X_reduced_train = pca.fit_transform(scale(X_train))
X_reduced_test = pca.transform(scale(X_test))[:,:5]

#train PCR model on training data
regr = LinearRegression()
regr.fit(X_reduced_train[:,:5], y_train)

#calculate RMSE
pred = regr.predict(X_reduced_test)
print('\033[1m','MSE: ','\033[0m',np.sqrt(mean_squared_error(y_test, pred)))
```

```
 MSE:   0.3033478080454169
```

```
Pruebar=pd.DataFrame(y_test)
Pruebar["Prediccion"]=pred
Pruebar
```

	diabetes	Prediccion
54	0	-0.053377
124	0	-0.023556
349	0	0.180013

```
Pruebar.loc[Pruebar['Prediccion'] > 0.3, 'Prediccion'] = 1
Pruebar.loc[Pruebar['Prediccion'] < 0.3, 'Prediccion'] = 0
```

```
frecp=Pruebar['Prediccion'].value_counts()
frecp
```

```
0.0    92
1.0    25
Name: Prediccion, dtype: int64
```

```
frecr=Pruebar['diabetes'].value_counts()
frecr
```

```
0    95
1    22
Name: diabetes, dtype: int64
```

```
print("Accuracy :",frecp[0]/(frecr[0]+frecr[1]))
```

```
Accuracy : 0.7863247863247863
```

Regresión lineal

```
df_regresion
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_c
0	193	77	49	3.9	19	female	61	119	22.5	118	70	32	38	
1	146	79	41	3.6	19	female	60	135	26.4	108	58	33	40	
2	217	75	54	4.0	20	female	67	187	29.3	110	72	40	45	
3	226	97	70	3.2	20	female	64	114	19.6	122	64	31	39	
4	164	91	67	2.4	20	female	70	141	20.2	122	86	32	39	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
385	227	105	44	5.2	83	female	59	125	25.2	150	90	35	40	
386	226	279	52	4.3	84	female	60	192	37.5	144	88	41	48	
387	301	90	118	2.6	89	female	61	115	21.7	218	90	31	41	
388	232	184	114	2.0	91	female	61	127	24.0	170	82	35	38	
389	165	94	69	2.4	92	female	62	217	39.7	160	82	51	51	

390 rows x 15 columns

```
df_regresion.loc[df_regresion.diabetes == 'Diabetes','diabetes'] = 1
df_regresion.loc[df_regresion.diabetes == 'No_diabetes','diabetes'] = 0
```

```
df_regresion.loc[df_regresion.genero == 'female','genero'] = 0
df_regresion.loc[df_regresion.genero == 'male','genero'] = 1
```

```
!pip uninstall scipy -y
!pip uninstall pingouin -y
!pip install pingouin
```

```
Found existing installation: scipy 1.7.3
Uninstalling scipy-1.7.3:
  Successfully uninstalled scipy-1.7.3
WARNING: Skipping pingouin as it is not installed.
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pingouin
  Downloading pingouin-0.5.2.tar.gz (185 kB)
     |████████████████████████████████| 185 kB 5.1 MB/s
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.7/dist-packages (from pingouin) (1.21.6)
Collecting scipy>=1.7
  Downloading scipy-1.7.3-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (38.1 MB)
     |████████████████████████████████| 38.1 MB 1.3 MB/s
Requirement already satisfied: pandas>=1.0 in /usr/local/lib/python3.7/dist-packages (from pingouin) (1.3.5)
Requirement already satisfied: matplotlib>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from pingouin) (3.2.2)
Requirement already satisfied: seaborn>=0.11 in /usr/local/lib/python3.7/dist-packages (from pingouin) (0.11.2)
Collecting statsmodels>=0.13
  Downloading statsmodels-0.13.2-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (9.8 MB)
     |████████████████████████████████| 9.8 MB 38.4 MB/s
Requirement already satisfied: scikit-learn<1.1.0 in /usr/local/lib/python3.7/dist-packages (from pingouin) (1.0.2)
Collecting pandas_flavor>=0.2.0
  Downloading pandas_flavor-0.3.0-py3-none-any.whl (6.3 kB)
Collecting outdated
  Downloading outdated-0.2.1-py3-none-any.whl (7.5 kB)
Requirement already satisfied: tabulate in /usr/local/lib/python3.7/dist-packages (from pingouin) (0.8.10)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.2->pingouin) (0.11.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.2->pingouin) (1.4.4)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.2->pingouin) (2.8
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=3.0.2
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=1.0->pingouin) (2022.2.1)
Collecting pandas_flavor>=0.2.0
  Downloading pandas_flavor-0.2.0-py2.py3-none-any.whl (6.6 kB)
Requirement already satisfied: xarray in /usr/local/lib/python3.7/dist-packages (from pandas_flavor>=0.2.0->pingouin) (0.20.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=3.0.2->ping
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn<1.1.0->pingouin) (3.
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn<1.1.0->pingouin) (1.1.0)
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (from statsmodels>=0.13->pingouin) (0.5.2)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-packages (from statsmodels>=0.13->pingouin) (21.3)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from outdated->pingouin) (2.23.0)
Collecting littleutils
  Downloading littleutils-0.2.2.tar.gz (6.6 kB)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->outdated->pingouin) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->outdated->pingouin) (3.0.4
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->outdated->pingouin) (2022
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->outd
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from xarray->pandas_flavor>=0.2.0->pingo
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->xarray->pandas_flavor>=0
Building wheels for collected packages: pingouin, littleutils
  Building wheel for pingouin (setup.py) ... done
  Created wheel for pingouin: filename=pingouin-0.5.2-py3-none-any.whl size=196206 sha256=10e3ebf627281f573d1071be58133f7783ceb4b0d1d
  Stored in directory: /root/.cache/pip/wheels/11/5a/63/a6d32fc26fa462c731f65480bfb98ff7bd39b8ebcb4bc6c2fe
  Building wheel for littleutils (setup.py) ... done
  Created wheel for littleutils: filename=littleutils-0.2.2-py3-none-any.whl size=7048 sha256=fe5438283bb92b23678fce47d3fd84143fb8681
  Stored in directory: /root/.cache/pip/wheels/d6/64/cd/32819b511a488e4993f2fab909a95330289c3f4e0f6ef4676d
```

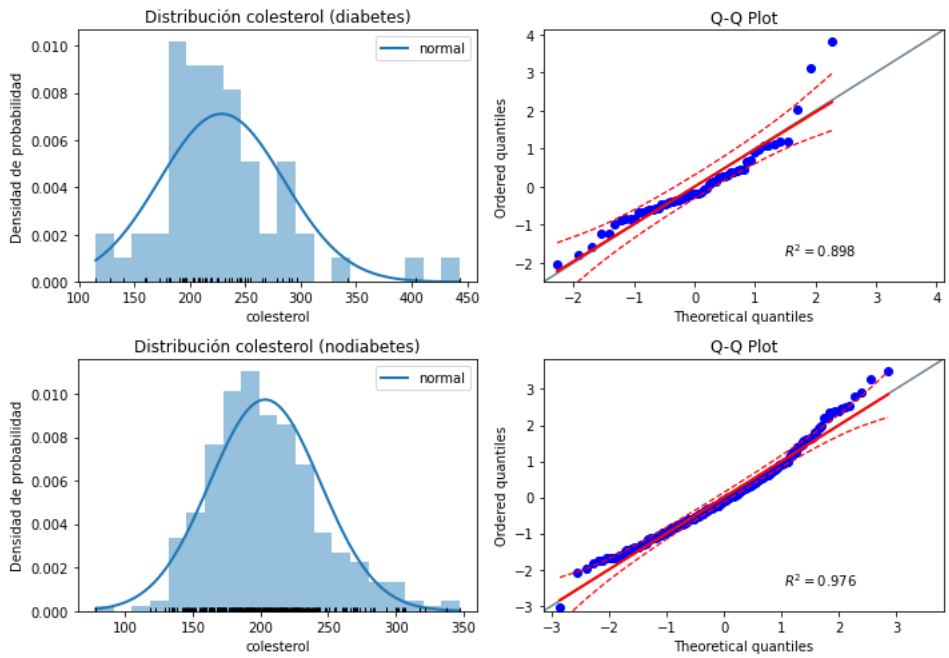
```
from scipy import stats
import pingouin as pg
```

```
df_regresion.groupby('diabetes').size()
```

```
diabetes
0    330
1     60
dtype: int64
```

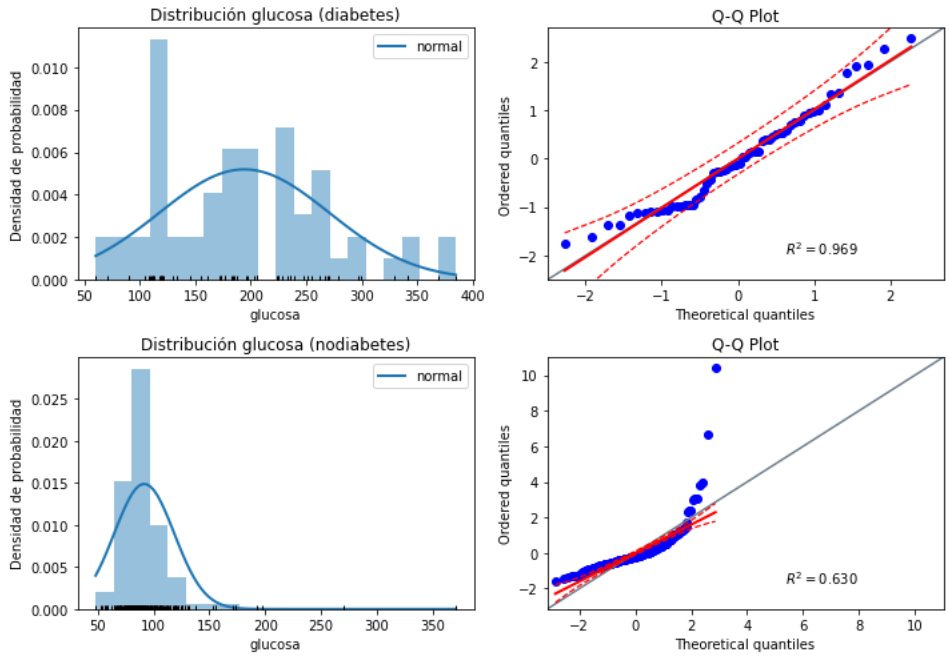
```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

colesterol_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'colesterol']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(colesterol_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(colesterol_diabetes), max(colesterol_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=colesterol_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(colesterol_diabetes, np.full_like(colesterol_diabetes, -0.01), '|k', markeredgewidth=1)
```

```
axs[0, 0].set_title('Distribución colesterol (diabetes)')
axs[0, 0].set_xlabel('colesterol')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(colesterol_diabetes, dist='norm', ax=axs[0, 1])

colesterol_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'colesterol']
mu, sigma = stats.norm.fit(colesterol_nodiabetes)
x_hat = np.linspace(min(colesterol_nodiabetes), max(colesterol_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=colesterol_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(colesterol_nodiabetes, np.full_like(colesterol_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución colesterol (nodiabetes)')
axs[1, 0].set_xlabel('colesterol')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(colesterol_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```



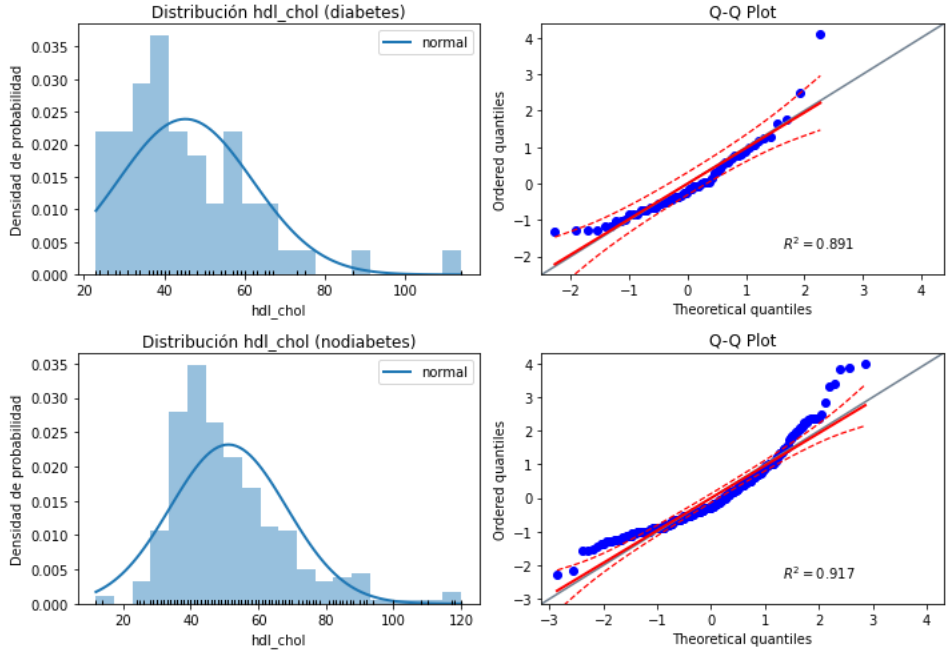

```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

glucosa_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'glucosa']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(glucosa_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(glucosa_diabetes), max(glucosa_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=glucosa_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(glucosa_diabetes, np.full_like(glucosa_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución glucosa (diabetes)')
axs[0, 0].set_xlabel('glucosa')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(glucosa_diabetes, dist='norm', ax=axs[0, 1])

glucosa_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'glucosa']
mu, sigma = stats.norm.fit(glucosa_nodiabetes)
```

```
x_hat = np.linspace(min(glucosa_nodiabetes), max(glucosa_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=glucosa_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(glucosa_nodiabetes, np.full_like(glucosa_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución glucosa (nodiabetes)')
axs[1, 0].set_xlabel('glucosa')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(glucosa_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```

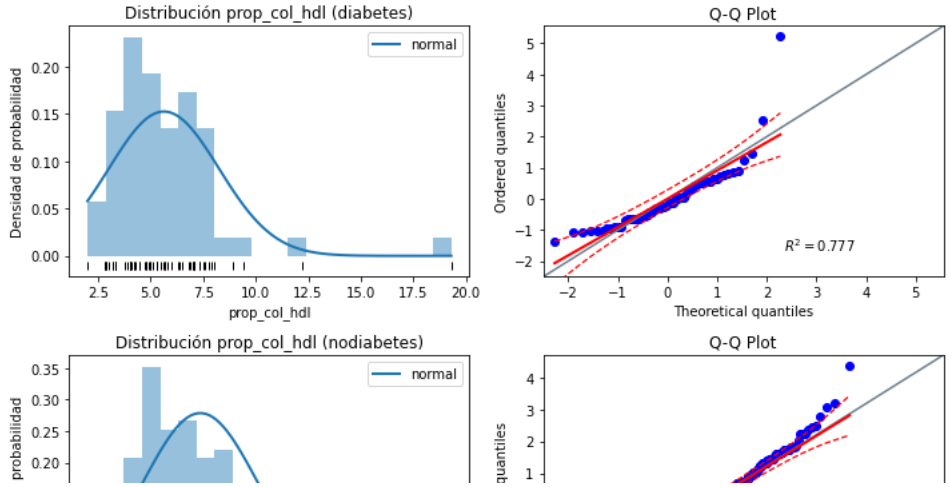



```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

hdl_chol_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'hdl_chol']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(hdl_chol_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(hdl_chol_diabetes), max(hdl_chol_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=hdl_chol_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(hdl_chol_diabetes, np.full_like(hdl_chol_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución hdl_chol (diabetes)')
axs[0, 0].set_xlabel('hdl_chol')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(hdl_chol_diabetes, dist='norm', ax=axs[0, 1])

hdl_chol_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'hdl_chol']
mu, sigma = stats.norm.fit(hdl_chol_nodiabetes)
x_hat = np.linspace(min(hdl_chol_nodiabetes), max(hdl_chol_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=hdl_chol_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(hdl_chol_nodiabetes, np.full_like(hdl_chol_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución hdl_chol (nodiabetes)')
axs[1, 0].set_xlabel('hdl_chol')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

```

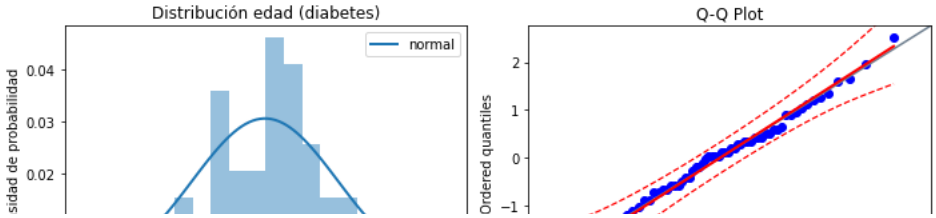
```
pg.qqplot(hdl_chol_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```



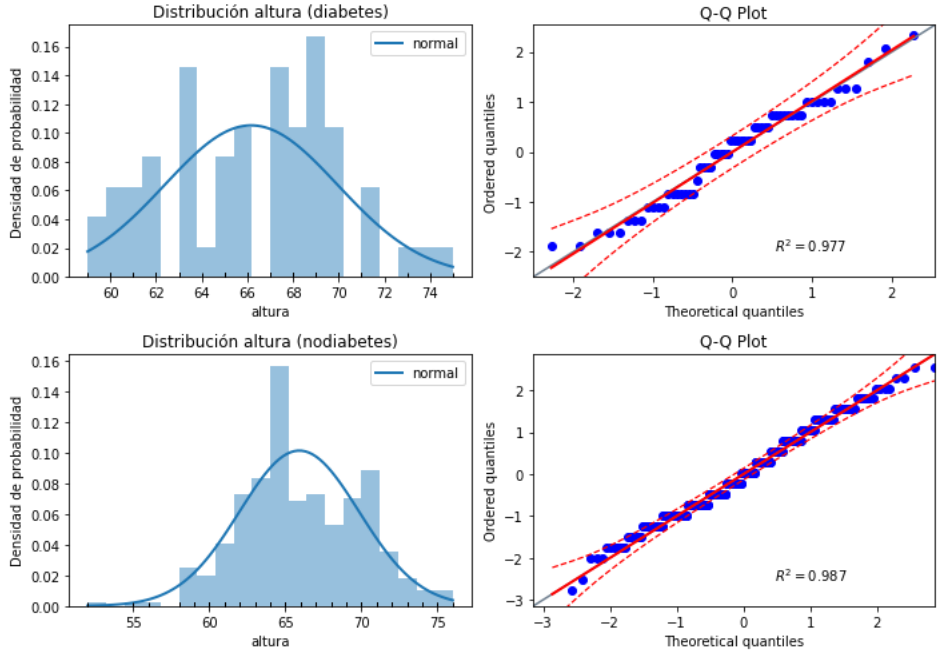

```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

prop_col_hdl_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'prop_col_hdl']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(prop_col_hdl_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(prop_col_hdl_diabetes), max(prop_col_hdl_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=prop_col_hdl_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(prop_col_hdl_diabetes, np.full_like(prop_col_hdl_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución prop_col_hdl (diabetes)')
axs[0, 0].set_xlabel('prop_col_hdl')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(prop_col_hdl_diabetes, dist='norm', ax=axs[0, 1])

prop_col_hdl_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'prop_col_hdl']
mu, sigma = stats.norm.fit(prop_col_hdl_nodiabetes)
x_hat = np.linspace(min(prop_col_hdl_nodiabetes), max(prop_col_hdl_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=prop_col_hdl_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(prop_col_hdl_nodiabetes, np.full_like(prop_col_hdl_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución prop_col_hdl (nodiabetes)')
axs[1, 0].set_xlabel('prop_col_hdl')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(prop_col_hdl_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```



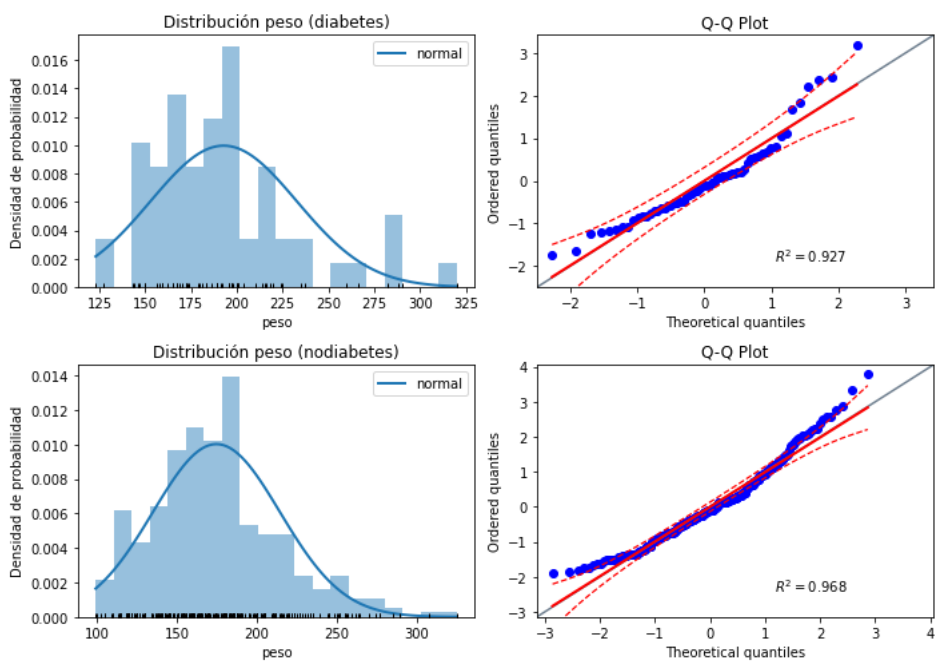

```
# Gráficos de distribución
# ==========================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

edad_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'edad']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(edad_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(edad_diabetes), max(edad_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=edad_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(edad_diabetes, np.full_like(edad_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución edad (diabetes)')
axs[0, 0].set_xlabel('edad')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(edad_diabetes, dist='norm', ax=axs[0, 1])

edad_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'edad']
mu, sigma = stats.norm.fit(edad_nodiabetes)
x_hat = np.linspace(min(edad_nodiabetes), max(edad_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=edad_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(edad_nodiabetes, np.full_like(edad_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución edad (nodiabetes)')
axs[1, 0].set_xlabel('edad')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(edad_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```




```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

altura_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'altura']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(altura_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(altura_diabetes), max(altura_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=altura_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(altura_diabetes, np.full_like(altura_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución altura (diabetes)')
axs[0, 0].set_xlabel('altura')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(altura_diabetes, dist='norm', ax=axs[0, 1])

altura_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'altura']
mu, sigma = stats.norm.fit(altura_nodiabetes)
x_hat = np.linspace(min(altura_nodiabetes), max(altura_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=altura_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(altura_nodiabetes, np.full_like(altura_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución altura (nodiabetes)')
axs[1, 0].set_xlabel('altura')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(altura_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```

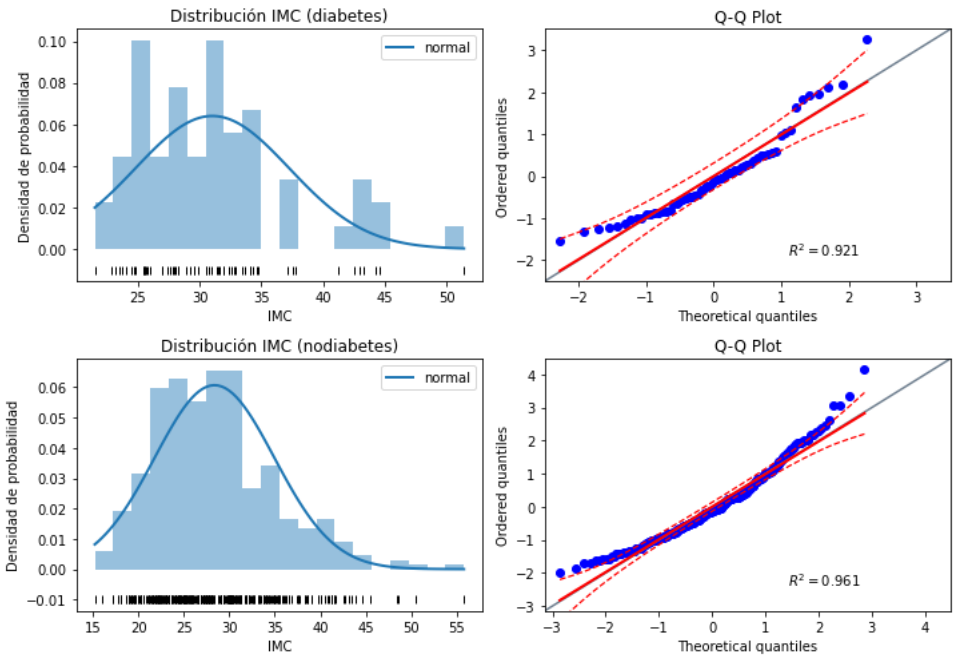


```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

peso_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'peso']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(peso_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(peso_diabetes), max(peso_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=peso_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(peso_diabetes, np.full_like(peso_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución peso (diabetes)')
axs[0, 0].set_xlabel('peso')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(peso_diabetes, dist='norm', ax=axs[0, 1])

peso_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'peso']
mu, sigma = stats.norm.fit(peso_nodiabetes)
x_hat = np.linspace(min(peso_nodiabetes), max(peso_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=peso_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(peso_nodiabetes, np.full_like(peso_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución peso (nodiabetes)')
axs[1, 0].set_xlabel('peso')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(peso_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```



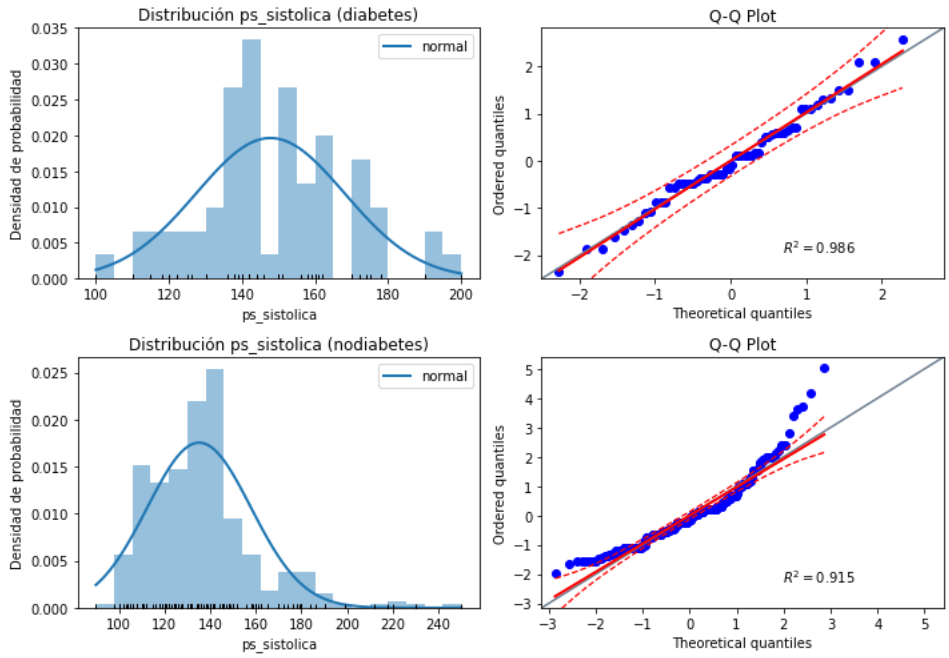

```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

IMC_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'IMC']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(IMC_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(IMC_diabetes), max(IMC_diabetes), num=100)
```

```
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=IMC_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(IMC_diabetes, np.full_like(IMC_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución IMC (diabetes)')
axs[0, 0].set_xlabel('IMC')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(IMC_diabetes, dist='norm', ax=axs[0, 1])

IMC_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'IMC']
mu, sigma = stats.norm.fit(IMC_nodiabetes)
x_hat = np.linspace(min(IMC_nodiabetes), max(IMC_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=IMC_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(IMC_nodiabetes, np.full_like(IMC_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución IMC (nodiabetes)')
axs[1, 0].set_xlabel('IMC')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(IMC_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```



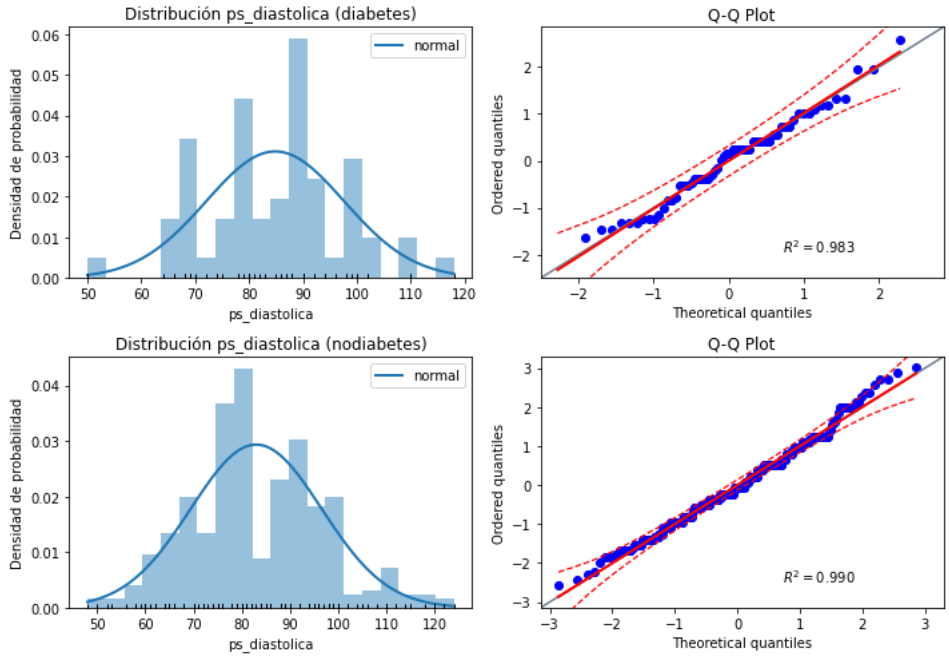

```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

ps_sistolica_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'ps_sistolica']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(ps_sistolica_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(ps_sistolica_diabetes), max(ps_sistolica_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=ps_sistolica_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(ps_sistolica_diabetes, np.full_like(ps_sistolica_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución ps_sistolica (diabetes)')
axs[0, 0].set_xlabel('ps_sistolica')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()
```

```

# Gráfico distribución qq-plot
pg.qqplot(ps_sistolica_diabetes, dist='norm', ax=axs[0, 1])

ps_sistolica_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'ps_sistolica']
mu, sigma = stats.norm.fit(ps_sistolica_nodiabetes)
x_hat = np.linspace(min(ps_sistolica_nodiabetes), max(ps_sistolica_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=ps_sistolica_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(ps_sistolica_nodiabetes, np.full_like(ps_sistolica_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución ps_sistolica (nodiabetes)')
axs[1, 0].set_xlabel('ps_sistolica')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(ps_sistolica_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```

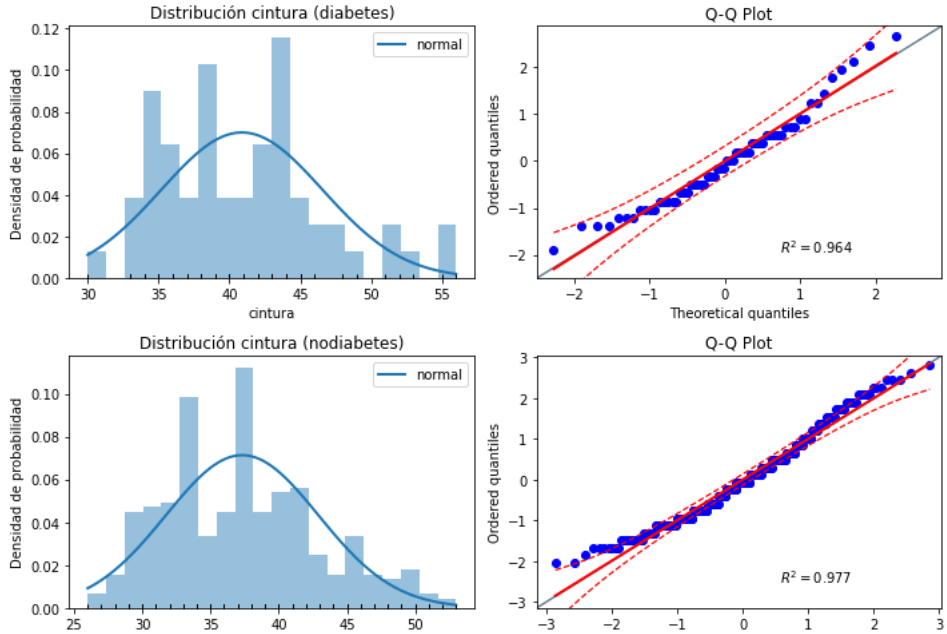



```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

ps_diastolica_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'ps_diastolica']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(ps_diastolica_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(ps_diastolica_diabetes), max(ps_diastolica_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=ps_diastolica_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(ps_diastolica_diabetes, np.full_like(ps_diastolica_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución ps_diastolica (diabetes)')
axs[0, 0].set_xlabel('ps_diastolica')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(ps_diastolica_diabetes, dist='norm', ax=axs[0, 1])

ps_diastolica_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'ps_diastolica']
mu, sigma = stats.norm.fit(ps_diastolica_nodiabetes)
x_hat = np.linspace(min(ps_diastolica_nodiabetes), max(ps_diastolica_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=ps_diastolica_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
```

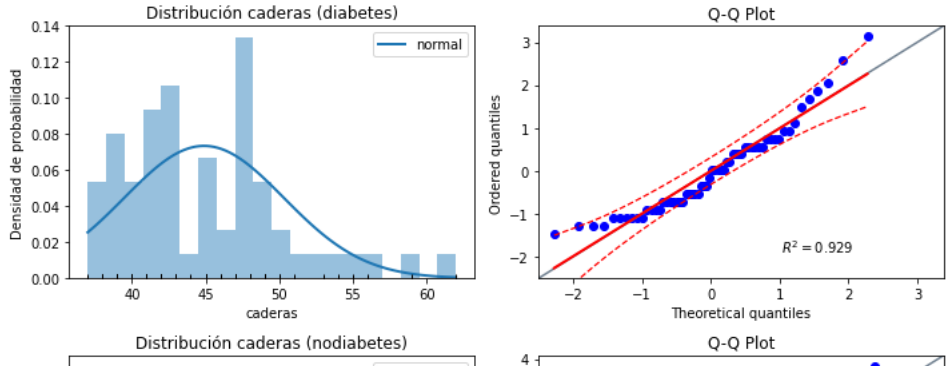
```
axs[1, 0].plot(ps_diastolica_nodiabetes, np.full_like(ps_diastolica_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución ps_diastolica (nodiabetes)')
axs[1, 0].set_xlabel('ps_diastolica')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(ps_diastolica_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```




```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

cintura_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'cintura']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(cintura_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(cintura_diabetes), max(cintura_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=cintura_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(cintura_diabetes, np.full_like(cintura_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución cintura (diabetes)')
axs[0, 0].set_xlabel('cintura')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(cintura_diabetes, dist='norm', ax=axs[0, 1])

cintura_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'cintura']
mu, sigma = stats.norm.fit(cintura_nodiabetes)
x_hat = np.linspace(min(cintura_nodiabetes), max(cintura_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=cintura_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(cintura_nodiabetes, np.full_like(cintura_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución cintura (nodiabetes)')
axs[1, 0].set_xlabel('cintura')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(cintura_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```




```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

caderas_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'caderas']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(caderas_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(caderas_diabetes), max(caderas_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=caderas_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(caderas_diabetes, np.full_like(caderas_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución caderas (diabetes)')
axs[0, 0].set_xlabel('caderas')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(caderas_diabetes, dist='norm', ax=axs[0, 1])

caderas_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'caderas']
mu, sigma = stats.norm.fit(caderas_nodiabetes)
x_hat = np.linspace(min(caderas_nodiabetes), max(caderas_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=caderas_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(caderas_nodiabetes, np.full_like(caderas_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución caderas (nodiabetes)')
axs[1, 0].set_xlabel('caderas')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(caderas_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```



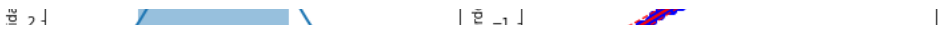

```
# Gráficos de distribución
# ==============================================================================
fig, axs = plt.subplots(2, 2, figsize=(10, 7))

prop_cin_cad_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'prop_cin_cad']
# Valores de la media (mu) y desviación típica (sigma) de cada grupo
mu, sigma = stats.norm.fit(prop_cin_cad_diabetes)

# Valores teóricos de la normal en el rango observado
x_hat = np.linspace(min(prop_cin_cad_diabetes), max(prop_cin_cad_diabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)

# Gráfico distribución
axs[0, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[0, 0].hist(x=prop_cin_cad_diabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[0, 0].plot(prop_cin_cad_diabetes, np.full_like(prop_cin_cad_diabetes, -0.01), '|k', markeredgewidth=1)
axs[0, 0].set_title('Distribución prop_cin_cad (diabetes)')
axs[0, 0].set_xlabel('prop_cin_cad')
axs[0, 0].set_ylabel('Densidad de probabilidad')
axs[0, 0].legend()

# Gráfico distribución qq-plot
pg.qqplot(prop_cin_cad_diabetes, dist='norm', ax=axs[0, 1])

prop_cin_cad_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'prop_cin_cad']
mu, sigma = stats.norm.fit(prop_cin_cad_nodiabetes)
x_hat = np.linspace(min(prop_cin_cad_nodiabetes), max(prop_cin_cad_nodiabetes), num=100)
y_hat = stats.norm.pdf(x_hat, mu, sigma)
axs[1, 0].plot(x_hat, y_hat, linewidth=2, label='normal')
axs[1, 0].hist(x=prop_cin_cad_nodiabetes, density=True, bins=20, color="#3182bd", alpha=0.5)
axs[1, 0].plot(prop_cin_cad_nodiabetes, np.full_like(prop_cin_cad_nodiabetes, -0.01), '|k', markeredgewidth=1)
axs[1, 0].set_title('Distribución prop_cin_cad (nodiabetes)')
axs[1, 0].set_xlabel('prop_cin_cad')
axs[1, 0].set_ylabel('Densidad de probabilidad')
axs[1, 0].legend()

pg.qqplot(prop_cin_cad_nodiabetes, dist='norm', ax=axs[1, 1])
plt.tight_layout();
```


```
pg.normality(data = df_regresion, dv = 'colesterol', group = 'diabetes')
```

	W	pval	normal
0	0.977382	0.000047	False
1	0.909198	0.000290	False


```
pg.normality(data = df_regresion, dv = 'colesterol', group = 'diabetes')
```

	W	pval	normal
0	0.977382	0.000047	False
1	0.909198	0.000290	False



```
pg.normality(data = df_regresion, dv = 'hdl_chol', group = 'diabetes')
```

	W	pval	normal
0	0.918263	1.999520e-12	False
1	0.898020	1.114847e-04	False

```
pg.normality(data = df_regresion, dv = 'prop_col_hdl', group = 'diabetes')
```

	W	pval	normal
0	0.953534	1.038987e-08	False
1	0.794430	9.833369e-08	False

```
pg.normality(data = df_regresion, dv = 'edad', group = 'diabetes')
```

	W	pval	normal
0	0.964710	3.585699e-07	False
1	0.988666	8.517852e-01	True

```
pg.normality(data = df_regresion, dv = 'altura', group = 'diabetes')
```

	W	pval	normal
0	0.987116	0.004932	False
1	0.972026	0.183158	True

```
pg.normality(data = df_regresion, dv = 'peso', group = 'diabetes')
```

	W	pval	normal
0	0.967653	0.000001	False
1	0.929339	0.001862	False

```
pg.normality(data = df_regresion, dv = 'IMC', group = 'diabetes')
```

	W	pval	normal
0	0.961125	1.081305e-07	False
1	0.921511	8.826908e-04	False

```
pg.normality(data = df_regresion, dv = 'ps_sistolica', group = 'diabetes')
```

	W	pval	normal
0	0.917309	1.647114e-12	False
1	0.987113	7.785110e-01	True

```
pg.normality(data = df_regresion, dv = 'ps_diastolica', group = 'diabetes')
```

	W	pval	normal
0	0.989816	0.021375	False
1	0.986149	0.729652	True

```
pg.normality(data = df_regresion, dv = 'cintura', group = 'diabetes')
```

	W	pval	normal
0	0.975017	0.000017	False
1	0.961822	0.057783	True

```
pg.normality(data = df_regresion, dv = 'caderas', group = 'diabetes')
```

	W	pval	normal
0	0.959707	6.847286e-08	False
1	0.928343	1.690424e-03	False

```
pg.normality(data = df_regresion, dv = 'prop_cin_cad', group = 'diabetes')
```

	W	pval	normal
0	0.990321	0.028313	False
1	0.966866	0.102353	True

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
colesterol_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'colesterol']
colesterol_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'colesterol']

pg.ttest(x=colesterol_diabetes, y = colesterol_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	4.110156	388	two-sided	0.000048	[13.17, 37.34]	0.576843	383.017	0.983823

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
glucosa_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'glucosa']
glucosa_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'glucosa']

pg.ttest(x=glucosa_diabetes, y=glucosa_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	18.729901	388	two-sided	3.205119e-56	[91.84, 113.39]	2.628665	2.72e+52	1.0

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
hdl_chol_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'hdl_chol']
hdl_chol_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'hdl_chol']

pg.ttest(x=hdl_chol_diabetes, y=hdl_chol_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	-2.444045	388	two-sided	0.014968	[-10.63, -1.15]	0.343012	2.479	0.68369

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
prop_col_hdl_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'prop_col_hdl']
prop_col_hdl_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'prop_col_hdl']

pg.ttest(x=prop_col_hdl_diabetes, y=prop_col_hdl_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	5.589515	388	two-sided	4.298115e-08	[0.85, 1.77]	0.784465	2.522e+05	0.99985

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
edad_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'edad']
edad_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'edad']

pg.ttest(x=edad_diabetes, y=edad_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	6.239951	388	two-sided	1.146990e-09	[9.41, 18.07]	0.875751	7.643e+06	1.0

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
altura_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'altura']
altura_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'altura']

pg.ttest(x=altura_diabetes, y=altura_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	0.462345	388	two-sided	0.644093	[-0.83, 1.34]	0.064888	0.169	0.074703

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
peso_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'peso']
peso_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'peso']

pg.ttest(x=peso_diabetes, y=peso_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	3.25399	388	two-sided	0.001238	[7.22, 29.25]	0.456684	21.063	0.900772

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
IMC = df_regresion.loc[df_regresion.diabetes == 1, 'IMC']
IMC_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'IMC']

pg.ttest(x=IMC_diabetes, y=IMC_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	2.892241	388	two-sided	0.004041	[0.85, 4.46]	0.405914	7.526	0.822546

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
ps_sistolica_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'ps_sistolica']
ps_sistolica_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'ps_sistolica']

pg.ttest(x=ps_sistolica_diabetes, y=ps_sistolica_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	3.991465	388	two-sided	0.000079	[6.38, 18.76]	0.560186	246.536	0.978391

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
ps_diastolica_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'ps_diastolica']
ps_diastolica_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'ps_diastolica']
```

```
pg.ttest(x=ps_diastolica_diabetes, y=ps_diastolica_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	0.973289	388	two-sided	0.331016	[-1.88, 5.57]	0.136597	0.238	0.163002

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
cintura_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'cintura']
cintura_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'cintura']

pg.ttest(x=cintura_diabetes, y=cintura_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	4.514098	388	two-sided	0.000008	[2.01, 5.11]	0.633535	1878.979	0.994504

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
caderas_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'caderas']
caderas_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'caderas']

pg.ttest(x=caderas_diabetes, y=caderas_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	2.862119	388	two-sided	0.004436	[0.71, 3.8]	0.401687	6.944	0.814624

```
# Test para datos independientes (p-value, intervalos de confianza)
# ==========================================================================
prop_cin_cad_diabetes = df_regresion.loc[df_regresion.diabetes == 1, 'prop_cin_cad']
prop_cin_cad_nodiabetes = df_regresion.loc[df_regresion.diabetes == 0, 'prop_cin_cad']

pg.ttest(x=prop_cin_cad_diabetes, y=prop_cin_cad_nodiabetes, alternative='two-sided', correction=False)
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	3.513984	388	two-sided	0.000494	[0.02, 0.06]	0.493173	47.454	0.938864

Regresión con variables significativas

```
df_regresion.head()
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_cin_
0	193	77	49	3.9	19	0	61	119	22.5	118	70	32	38	
1	146	79	41	3.6	19	0	60	135	26.4	108	58	33	40	
2	217	75	54	4.0	20	0	67	187	29.3	110	72	40	45	
3	226	97	70	3.2	20	0	64	114	19.6	122	64	31	39	
4	164	91	67	2.4	20	0	70	141	20.2	122	86	32	39	

```
df1 = df_regresion.drop(["altura"], axis = 1)
df2 = df1.drop(["ps_diastolica"], axis = 1)
df2
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	peso	IMC	ps_sistolica	cintura	caderas	prop_cin_cad	diabetes
0	193	77	49	3.9	19	0	119	22.5	118	32	38	0.84	0
1	146	79	41	3.6	19	0	135	26.4	108	33	40	0.83	0
2	217	75	54	4.0	20	0	187	29.3	110	40	45	0.89	0
3	226	97	70	3.2	20	0	114	19.6	122	31	39	0.79	0
4	164	91	67	2.4	20	0	141	20.2	122	32	39	0.82	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
385	227	105	44	5.2	83	0	125	25.2	150	35	40	0.88	0

```
df2 = df2.drop(["genero"], axis = 1)
```

```
df3 = df2.drop(['colesterol'], axis=1)
df3 = df3.drop(['glucosa'], axis =1)
df3 = df3.drop(['hdl_chol'], axis =1)
df3 = df3.drop(['prop_col_hdl'], axis =1)
df3 = df3.drop(['edad'], axis =1)
df3 = df3.drop(['peso'], axis =1)
df3 = df3.drop(['IMC'], axis =1)
df3 = df3.drop(['ps_sistolica'], axis =1)
df3 = df3.drop(['cintura'], axis =1)
df3 = df3.drop(['caderas'], axis =1)
Y = df3.drop(['prop_cin_cad'], axis =1)

Y
```

	diabetes
0	0
1	0
2	0
3	0
4	0
...	...
385	0
386	1
387	0
388	1
389	0

390 rows × 1 columns

```
dfx = df2.drop(['diabetes'], axis=1)
dfx
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	peso	IMC	ps_sistolica	cintura	caderas	prop_cin_cad
1	146	79	41	3.6	19	135	26.4	108	33	40	0.83
3	226	97	70	3.2	20	114	19.6	122	31	39	0.79

```
1 from numpy import *
```

```
1 Xn = ones(390)
```

```
1 dfx['Xn'] = Xn
2 X = dfx
3 X
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	peso	IMC	ps_sistolica	cintura	caderas	prop_cin_cad	Xn
0	193	77	49	3.9	19	119	22.5	118	32	38	0.84	1.0
1	146	79	41	3.6	19	135	26.4	108	33	40	0.83	1.0
2	217	75	54	4.0	20	187	29.3	110	40	45	0.89	1.0
3	226	97	70	3.2	20	114	19.6	122	31	39	0.79	1.0
4	164	91	67	2.4	20	141	20.2	122	32	39	0.82	1.0
...	...	...	...	...	...	...	...	...	...	...	...	...
385	227	105	44	5.2	83	125	25.2	150	35	40	0.88	1.0
386	226	279	52	4.3	84	192	37.5	144	41	48	0.85	1.0
387	301	90	118	2.6	89	115	21.7	218	31	41	0.76	1.0
388	232	184	114	2.0	91	127	24.0	170	35	38	0.92	1.0
389	165	94	69	2.4	92	217	39.7	160	51	51	1.00	1.0

390 rows × 12 columns

```
1 Xt = X.transpose()
2 Xt
```

	0	1	2	3	4	5	6	7	8	9	...	380	
colesterol	193.00	146.00	217.00	226.00	164.00	170.00	149.00	164.00	230.00	179.00	...	157.00	252
glucosa	77.00	79.00	75.00	97.00	91.00	69.00	77.00	71.00	112.00	105.00	...	92.00	161
hdl_chol	49.00	41.00	54.00	70.00	67.00	64.00	49.00	63.00	64.00	60.00	...	47.00	87
prop_col_hdl	3.90	3.60	4.00	3.20	2.40	2.70	3.00	2.60	3.60	3.00	...	3.30	2
edad	19.00	19.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	20.00	...	80.00	80
peso	119.00	135.00	187.00	114.00	141.00	161.00	115.00	145.00	159.00	170.00	...	212.00	162
IMC	22.50	26.40	29.30	19.60	20.20	27.60	21.00	19.70	24.90	35.50	...	29.60	29
ps_sistolica	118.00	108.00	110.00	122.00	122.00	108.00	105.00	108.00	100.00	140.00	...	156.00	160
cintura	32.00	33.00	40.00	31.00	32.00	37.00	31.00	29.00	31.00	34.00	...	47.00	44
caderas	38.00	40.00	45.00	39.00	39.00	40.00	37.00	36.00	39.00	46.00	...	48.00	41
prop_cin_cad	0.84	0.83	0.89	0.79	0.82	0.93	0.84	0.81	0.79	0.74	...	0.98	1
Xn	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	...	1.00	1

12 rows × 390 columns

```
1 X = X.to_numpy()
2 X
```

```
array([[193.  ,  77.  ,  49.  , ...,  38.  ,   0.84,   1.  ],
       [146.  ,  79.  ,  41.  , ...,  40.  ,   0.83,   1.  ],
       [217.  ,  75.  ,  54.  , ...,  45.  ,   0.89,   1.  ],
       ...,
       [301.  ,  90.  , 118.  , ...,  41.  ,   0.76,   1.  ],
       [232.  , 184.  , 114.  , ...,  38.  ,   0.92,   1.  ],
       [165.  ,  94.  ,  69.  , ...,  51.  ,   1.  ,   1.  ]])
```

```
Xt = Xt.to_numpy()
Xt
```

```
array([[193.  , 146.  , 217.  , ..., 301.  , 232.  , 165.  ],
       [ 77.  ,  79.  ,  75.  , ...,  90.  , 184.  ,  94.  ],
       [ 49.  ,  41.  ,  54.  , ..., 118.  , 114.  ,  69.  ],
       ...,
       [ 38.  ,  40.  ,  45.  , ...,  41.  ,  38.  ,  51.  ],
       [  0.84,   0.83,   0.89, ...,   0.76,   0.92,   1.  ],
       [  1.  ,   1.  ,   1.  , ...,   1.  ,   1.  ,   1.  ]])
```

```
A = Xt@X
print(A)
```

```
[[1.7524466e+07 8.8228800e+06 4.1205440e+06 3.8004010e+05 3.8509360e+06
  1.4381871e+07 2.3361639e+06 1.1165628e+07 3.0740080e+06 3.4838270e+06
  7.1350340e+04 8.0820000e+04]
 [8.8228800e+06 5.6192640e+06 2.0470200e+06 1.9966590e+05 2.0593280e+06
  7.5876140e+06 1.2224656e+06 5.8185470e+06 1.6120870e+06 1.8161290e+06
  3.7180150e+04 4.1862000e+04]
 [4.1205440e+06 2.0470200e+06 1.1015700e+06 8.0741200e+04 9.2008100e+05
  3.3986240e+06 5.5338670e+05 2.6932490e+06 7.3167400e+05 8.3429900e+05
  1.7200530e+04 1.9604000e+04]
 [3.8004010e+05 1.9966590e+05 8.0741200e+04 9.1573200e+03 8.4350100e+04
  3.2066450e+05 5.1796020e+04 2.4376920e+05 6.8043200e+04 7.6663600e+04
  1.5673260e+03 1.7646000e+03]
 [3.8509360e+06 2.0593280e+06 9.2008100e+05 8.4350100e+04 9.5834200e+05
  3.2216010e+06 5.2453850e+05 2.5678550e+06 6.9635700e+05 7.8443500e+05
  1.6207030e+04 1.8242000e+04]
 [1.4381871e+07 7.5876140e+06 3.3986240e+06 3.2066450e+05 3.2216010e+06
  1.2909817e+07 2.0802045e+06 9.5231510e+06 2.6969030e+06 3.0482260e+06
  6.1270350e+04 6.9189000e+04]
 [2.3361639e+06 1.2224656e+06 5.5338670e+05 5.1796020e+04 5.2453850e+05
  2.0802045e+06 3.3988417e+05 1.5461052e+06 4.3697990e+05 4.9530560e+05
  9.9103020e+03 1.1222500e+04]
 [1.1165628e+07 5.8185470e+06 2.6932490e+06 2.4376920e+05 2.5678550e+06
  9.5231510e+06 1.5461052e+06 7.5374400e+06 2.0361280e+06 2.3071380e+06
  4.7227970e+04 5.3482000e+04]
 [3.0740080e+06 1.6120870e+06 7.3167400e+05 6.8043200e+04 6.9635700e+05
  2.6969030e+06 4.3697990e+05 2.0361280e+06 5.7220100e+05 6.4555500e+05
  1.3101530e+04 1.4769000e+04]
 [3.4838270e+06 1.8161290e+06 8.3429900e+05 7.6663600e+04 7.8443500e+05
  3.0482260e+06 4.9530560e+05 2.3071380e+06 6.4555500e+05 7.3333300e+05
  1.4772100e+04 1.6767000e+04]
 [7.1350340e+04 3.7180150e+04 1.7200530e+04 1.5673260e+03 1.6207030e+04
  6.1270350e+04 9.9103020e+03 4.7227970e+04 1.3101530e+04 1.4772100e+04
  3.0505220e+02 3.4374000e+02]
 [8.0820000e+04 4.1862000e+04 1.9604000e+04 1.7646000e+03 1.8242000e+04
  6.9189000e+04 1.1222500e+04 5.3482000e+04 1.4769000e+04 1.6767000e+04
  3.4374000e+02 3.9000000e+02]]
```

```
A = pd.DataFrame(A)
A
```

	0	1	2	3	4	5	6	7	8	9	10	
0	17524466.00	8822880.00	4120544.00	380040.100	3850936.00	14381871.00	2336163.900	11165628.00	3074008.00	3483827.0	71350.3400	8
1	8822880.00	5619264.00	2047020.00	199665.900	2059328.00	7587614.00	1222465.600	5818547.00	1612087.00	1816129.0	37180.1500	4
2	4120544.00	2047020.00	1101570.00	80741.200	920081.00	3398624.00	553386.700	2693249.00	731674.00	834299.0	17200.5300	1
3	380040.10	199665.90	80741.20	9157.320	84350.10	320664.50	51796.020	243769.20	68043.20	76663.6	1567.3260	
4	3850936.00	2059328.00	920081.00	84350.100	958342.00	3221601.00	524538.500	2567855.00	696357.00	784435.0	16207.0300	1
5	14381871.00	7587614.00	3398624.00	320664.500	3221601.00	12909817.00	2080204.500	9523151.00	2696903.00	3048226.0	61270.3500	6
6	2336163.90	1222465.60	553386.70	51796.020	524538.50	2080204.50	339884.170	1546105.20	436979.90	495305.6	9910.3020	1
7	11165628.00	5818547.00	2693249.00	243769.200	2567855.00	9523151.00	1546105.200	7537440.00	2036128.00	2307138.0	47227.9700	5
8	3074008.00	1612087.00	731674.00	68043.200	696357.00	2696903.00	436979.900	2036128.00	572201.00	645555.0	13101.5300	1
9	3483827.00	1816129.00	834299.00	76663.600	784435.00	3048226.00	495305.600	2307138.00	645555.00	733333.0	14772.1000	1
10	71350.34	37180.15	17200.53	1567.326	16207.03	61270.35	9910.302	47227.97	13101.53	14772.1	305.0522	
11	80820.00	41862.00	19604.00	1764.600	18242.00	69189.00	11222.500	53482.00	14769.00	16767.0	343.7400	

```
A = A.to_numpy()
A
```

```
array([[1.7524466e+07, 8.8228800e+06, 4.1205440e+06, 3.8004010e+05,
        3.8509360e+06, 1.4381871e+07, 2.3361639e+06, 1.1165628e+07,
        3.0740080e+06, 3.4838270e+06, 7.1350340e+04, 8.0820000e+04],
       [8.8228800e+06, 5.6192640e+06, 2.0470200e+06, 1.9966590e+05,
        2.0593280e+06, 7.5876140e+06, 1.2224656e+06, 5.8185470e+06,
        1.6120870e+06, 1.8161290e+06, 3.7180150e+04, 4.1862000e+04],
       [4.1205440e+06, 2.0470200e+06, 1.1015700e+06, 8.0741200e+04,
        9.2008100e+05, 3.3986240e+06, 5.5338670e+05, 2.6932490e+06,
        7.3167400e+05, 8.3429900e+05, 1.7200530e+04, 1.9604000e+04],
       [3.8004010e+05, 1.9966590e+05, 8.0741200e+04, 9.1573200e+03,
        8.4350100e+04, 3.2066450e+05, 5.1796020e+04, 2.4376920e+05,
        6.8043200e+04, 7.6663600e+04, 1.5673260e+03, 1.7646000e+03],
       [3.8509360e+06, 2.0593280e+06, 9.2008100e+05, 8.4350100e+04,
        9.5834200e+05, 3.2216010e+06, 5.2453850e+05, 2.5678550e+06,
        6.9635700e+05, 7.8443500e+05, 1.6207030e+04, 1.8242000e+04],
       [1.4381871e+07, 7.5876140e+06, 3.3986240e+06, 3.2066450e+05,
        3.2216010e+06, 1.2909817e+07, 2.0802045e+06, 9.5231510e+06,
        2.6969030e+06, 3.0482260e+06, 6.1270350e+04, 6.9189000e+04],
       [2.3361639e+06, 1.2224656e+06, 5.5338670e+05, 5.1796020e+04,
        5.2453850e+05, 2.0802045e+06, 3.3988417e+05, 1.5461052e+06,
        4.3697990e+05, 4.9530560e+05, 9.9103020e+03, 1.1222500e+04],
       [1.1165628e+07, 5.8185470e+06, 2.6932490e+06, 2.4376920e+05,
        2.5678550e+06, 9.5231510e+06, 1.5461052e+06, 7.5374400e+06,
        2.0361280e+06, 2.3071380e+06, 4.7227970e+04, 5.3482000e+04],
       [3.0740080e+06, 1.6120870e+06, 7.3167400e+05, 6.8043200e+04,
        6.9635700e+05, 2.6969030e+06, 4.3697990e+05, 2.0361280e+06,
        5.7220100e+05, 6.4555500e+05, 1.3101530e+04, 1.4769000e+04],
       [3.4838270e+06, 1.8161290e+06, 8.3429900e+05, 7.6663600e+04,
        7.8443500e+05, 3.0482260e+06, 4.9530560e+05, 2.3071380e+06,
        6.4555500e+05, 7.3333300e+05, 1.4772100e+04, 1.6767000e+04],
       [7.1350340e+04, 3.7180150e+04, 1.7200530e+04, 1.5673260e+03,
        1.6207030e+04, 6.1270350e+04, 9.9103020e+03, 4.7227970e+04,
        1.3101530e+04, 1.4772100e+04, 3.0505220e+02, 3.4374000e+02],
       [8.0820000e+04, 4.1862000e+04, 1.9604000e+04, 1.7646000e+03,
        1.8242000e+04, 6.9189000e+04, 1.1222500e+04, 5.3482000e+04,
        1.4769000e+04, 1.6767000e+04, 3.4374000e+02, 3.9000000e+02]])
```

```
Ainversa = np.linalg.inv(A)
Ainversa
```

```
array([[ 4.94430189e-06,  8.30875813e-08, -1.22794199e-05,
        -1.43097618e-04, -4.06099610e-07,  6.18196040e-07,
        -2.98639638e-06, -2.41166045e-07, -9.33860049e-06,
         4.75920174e-06,  4.46317211e-04,  1.51663302e-05],
       [ 8.30875813e-08,  1.07553607e-06, -2.64098832e-07,
        -9.13663609e-06, -9.72883975e-07, -3.97380290e-07,
         1.37210804e-06, -1.02102650e-08, -6.73618227e-06,
         5.74597225e-06,  2.84163462e-04, -2.42524205e-04],
       [-1.22794199e-05, -2.64098832e-07,  4.77307218e-05,
         4.71891155e-04, -9.05120278e-07, -9.70830515e-07,
         7.51892417e-06, -3.58922130e-07, -1.34130321e-05,
         2.16459407e-05,  5.31877151e-04, -2.80539720e-03],
       [-1.43097618e-04, -9.13663609e-06,  4.71891155e-04,
         5.94388341e-03, -9.07028984e-06, -2.14736135e-05,
         8.45551974e-05, -2.06786682e-08, -2.50053847e-05,
         1.18466966e-04, -3.24524961e-03, -1.94616133e-02],
       [-4.06099610e-07, -9.72883975e-07, -9.05120278e-07,
        -9.07028984e-06,  1.50549218e-05,  3.00758329e-06,
        -2.49867666e-06, -3.81868142e-06,  4.27629967e-05,
        -4.91626514e-05, -2.85862572e-03,  2.64670074e-03],
       [ 6.18196040e-07, -3.97380290e-07, -9.70830515e-07,
        -2.14736135e-05,  3.00758329e-06,  9.17431337e-06,
        -2.37450137e-05,  8.55723245e-08, -3.23892292e-05,
        -1.44509578e-06,  1.71470938e-04,  1.01328182e-04],
       [-2.98639638e-06,  1.37210804e-06,  7.51892417e-06,
         8.45551974e-05, -2.49867666e-06, -2.37450137e-05,
         3.55405554e-04,  1.22756200e-06,  1.18669169e-05,
        -2.36407289e-04, -1.24301512e-03,  4.45500033e-03],
       [-2.41166045e-07, -1.02102650e-08, -3.58922130e-07,
        -2.06786682e-08, -3.81868142e-06,  8.55723245e-08,
         1.22756200e-06,  6.49603885e-06, -2.40363445e-05,
         1.50409740e-05,  9.51180903e-04, -1.26826832e-03],
       [-9.33860049e-06, -6.73618227e-06, -1.34130321e-05,
        -2.50053847e-05,  4.27629967e-05, -3.23892292e-05,
         1.18669169e-05, -2.40363445e-05,  1.83010783e-02,
        -1.57246877e-02, -7.82434502e-01,  6.82764846e-01],
       [ 4.75920174e-06,  5.74597225e-06,  2.16459407e-05,
         1.18466966e-04, -4.91626514e-05, -1.44509578e-06,
        -2.36407289e-04,  1.50409740e-05, -1.57246877e-02,
         1.40072738e-02,  6.80852183e-01, -6.02746866e-01],
```

```
       [ 4.46317211e-04,  2.84163462e-04,  5.31877151e-04,
        -3.24524961e-03, -2.85862572e-03,  1.71470938e-04,
        -1.24301512e-03,  9.51180903e-04, -7.82434502e-01,
         6.80852183e-01,  3.42189389e+01, -2.99276846e+01],
       [ 1.51663302e-05, -2.42524205e-04, -2.80539720e-03,
        -1.94616133e-02,  2.64670074e-03,  1.01328182e-04,
         4.45500033e-03, -1.26826832e-03,  6.82764846e-01,
        -6.02746866e-01, -2.99276846e+01,  2.65939799e+01]])
```

```
Nueva = Ainversa@Xt
Nueva
```

```
array([[ 4.31608871e-05, -5.15180002e-05,  9.01256291e-05, ...,
        -1.47743852e-04, -3.15022041e-04, -1.99519131e-04],
       [ 1.31352433e-05,  1.72443476e-05, -7.29809082e-06, ...,
        -3.76445694e-05,  6.24762729e-05, -5.71001596e-05],
       [-1.82277581e-04, -8.71463511e-05, -1.32723083e-04, ...,
         1.10249133e-03,  1.43846040e-03,  4.68519758e-04],
       ...,
       [-3.51138200e-03,  4.53538241e-04,  1.13212097e-03, ...,
        -4.71751924e-05,  2.64130438e-03, -1.81948747e-02],
       [-1.77509031e-01,  2.42653868e-02,  4.51447764e-02, ...,
        -1.01895544e-01,  1.44400029e-01, -8.83122433e-01],
       [ 1.82634158e-01,  1.79443486e-02, -3.60063082e-02, ...,
        -2.94707724e-02, -2.01912801e-01,  7.26062396e-01]])
```

```
Y = Y.to_numpy()
Y
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
```

```
       [0],
       [0],
       [0],
       [0],
       [0],
```

```
ab = Nueva@Y
ab
```

```
array([[0.00030509457219941887],
       [0.004264917984844847],
       [0.00031474161544478097],
       [0.01218605325178176],
       [0.0015301827209237991],
       [-0.0006872416282639533],
       [0.004503259029726481],
       [0.0005624032704808533],
       [0.037401686754168995],
       [-0.030643187273626693],
       [-1.4564354572881797],
       [0.5902414894327777]], dtype=object)
```

```
ab = pd.DataFrame(ab)
ab
```

	0
0	0.000305
1	0.004265
2	0.000315
3	0.012186
4	0.00153
5	-0.000687
6	0.004503
7	0.000562
8	0.037402
9	-0.030643
10	-1.456435
11	0.590241

```
print("La fórmula del problema es: y =" ,ab.loc[11,0], "+",ab.loc[10,0],"x", "+",ab.loc[9,0],"x","+",ab.loc[8,0],"x","+",ab.loc[7,0],"x","
```

```
La fórmula del problema es: y = 0.5902414894327777 + -1.4564354572881797 x + -0.030643187273626693 x + 0.037401686754168995 x + 0.000562
```

```
print("Predicciones")
a = 0.5902414894327777 + -1.4564354572881797*df_regresion['prop_cin_cad'] + -0.030643187273626693*df_regresion['caderas'] + 0.037401686754
```

```
Predicciones
```

```
dfPredicciones = pd.DataFrame(a)
```

```
dfY = pd.DataFrame(Y)
```

```
dfY
dfY = dfY.rename(columns={ 0 :'Casos de Diabetes'})
```

```
freq = dfY['Casos de Diabetes'].value_counts()
freq
```

```
0    330
1     60
Name: Casos de Diabetes, dtype: int64
```

```
Comparativa = pd.concat([dfY, dfPredicciones], axis =1 )
```

```
Comparativa
```

	Casos de Diabetes	0
0	0	-0.035543
1	0	-0.055904
2	0	-0.041148
3	0	0.056836
4	0	-0.020507
...	...	...
385	0	0.225141
386	1	0.988941
387	0	0.208410
388	1	0.566372
389	0	0.241105

390 rows × 2 columns

```
Comparativa.columns
```

```
Index(['Casos de Diabetes', 0], dtype='object')
```

```
Comparativa = Comparativa.rename(columns={0:'Predicción'})
Comparativa.columns
```

```
Index(['Casos de Diabetes', 'Predicción'], dtype='object')
```

```
Comparativa
```

	Casos de Diabetes	Predicción
0	0	-0.035543
1	0	-0.055904
2	0	-0.041148
3	0	0.056836
4	0	-0.020507
...	...	...
385	0	0.225141
386	1	0.988941
387	0	0.208410
388	1	0.566372
389	0	0.241105

390 rows × 2 columns

```
Comparativa.loc[Comparativa['Predicción'] > 0.268, 'Predicciones'] = 'True'
Comparativa.loc[Comparativa['Predicción'] < 0.268, 'Predicciones'] = 'False'
```

```
Comparativa
```

	Casos de Diabetes	Predicción	Predicciones
0	0	-0.035543	False
1	0	-0.055904	False
2	0	-0.041148	False
3	0	0.056836	False
4	0	-0.020507	False

```
freq = Comparativa['Casos de Diabetes'].value_counts()
freq
```

```
0    330
1     60
Name: Casos de Diabetes, dtype: int64
```

```
freq = Comparativa['Predicciones'].value_counts()
freq
```

```
False    328
True      62
Name: Predicciones, dtype: int64
```

Regresion con variables más altas en "power"

```
df_regresion.head()
```

	colesterol	glucosa	hdl_chol	prop_col_hdl	edad	genero	altura	peso	IMC	ps_sistolica	ps_diastolica	cintura	caderas	prop_cin_
0	193	77	49	3.9	19	0	61	119	22.5	118	70	32	38	
1	146	79	41	3.6	19	0	60	135	26.4	108	58	33	40	
2	217	75	54	4.0	20	0	67	187	29.3	110	72	40	45	
3	226	97	70	3.2	20	0	64	114	19.6	122	64	31	39	
4	164	91	67	2.4	20	0	70	141	20.2	122	86	32	39	

```
df1 = df_regresion.drop(["altura","ps_diastolica","hdl_chol", "IMC", "caderas", "genero"], axis = 1)
df1
```

	colesterol	glucosa	prop_col_hdl	edad	peso	ps_sistolica	cintura	prop_cin_cad	diabetes
0	193	77	3.9	19	119	118	32	0.84	0
1	146	79	3.6	19	135	108	33	0.83	0
2	217	75	4.0	20	187	110	40	0.89	0
3	226	97	3.2	20	114	122	31	0.79	0
4	164	91	2.4	20	141	122	32	0.82	0
...	...	...	...	...	...	...	...	...	...
385	227	105	5.2	83	125	150	35	0.88	0
386	226	279	4.3	84	192	144	41	0.85	1
387	301	90	2.6	89	115	218	31	0.76	0
388	232	184	2.0	91	127	170	35	0.92	1
389	165	94	2.4	92	217	160	51	1.00	0

390 rows × 9 columns

```
df1,test = train_test_split(df1,test_size=.3, random_state = 20)
df1.shape
y_test = test['diabetes']
```

```
df2 = df1.drop(['colesterol'], axis=1)
df2 = df2.drop(['glucosa'], axis =1)
df2 = df2.drop(['prop_col_hdl'], axis =1)
df2 = df2.drop(['edad'], axis =1)
df2 = df2.drop(['peso'], axis =1)
df2 = df2.drop(['ps_sistolica'], axis =1)
df2 = df2.drop(['cintura'], axis =1)
Y = df2.drop(['prop_cin_cad'], axis =1)

Y
```

	diabetes
33	0
84	0
87	0
222	0
382	0
...	...
331	0
218	0
223	1
271	1
355	0

273 rows × 1 columns

```
dfx = df1.drop(['diabetes'], axis=1)
```

```
from numpy import *
```

```
Xn = ones(273)
```

```
dfx['Xn'] = Xn
X = dfx
X
```

	colesterol	glucosa	prop_col_hdl	edad	peso	ps_sistolica	cintura	prop_cin_cad	Xn
33	169	104	2.9	25	154	140	40	0.95	1.0
84	179	85	3.4	32	179	140	37	0.79	1.0
87	213	83	4.5	33	157	130	37	0.90	1.0
222	189	75	2.6	49	205	120	40	0.82	1.0
382	271	121	6.8	81	158	146	36	0.84	1.0
...	...	...	...	...	...	...	...	...	...
331	170	67	5.2	65	182	140	42	1.08	1.0
218	209	87	6.1	48	121	111	32	0.84	1.0
223	160	196	4.8	49	266	150	49	1.09	1.0
271	219	173	7.1	56	197	100	41	0.82	1.0
355	199	85	3.4	71	171	136	38	0.95	1.0

273 rows × 9 columns

```
Xt = X.transpose()
Xt
```

	33	84	87	222	382	383	320	171	16	311	...	40	118	162	71	278	331	
colesterol	169.00	179.00	213.0	189.00	271.00	240.00	283.00	172.00	244.00	196.0	...	220.00	227.00	179.00	195.00	201.00	170.00	2
glucosa	104.00	85.00	83.0	75.00	121.00	88.00	145.00	101.00	89.00	206.0	...	60.00	75.00	80.00	92.00	106.00	67.00	
prop_col_hdl	2.90	3.40	4.5	2.60	6.80	4.90	7.30	3.70	2.70	4.8	...	3.30	5.20	1.90	4.80	3.80	5.20	
edad	25.00	32.00	33.0	49.00	81.00	82.00	63.00	42.00	21.00	62.0	...	26.00	37.00	41.00	30.00	58.00	65.00	
peso	154.00	179.00	157.0	205.00	158.00	170.00	200.00	165.00	163.00	196.0	...	150.00	170.00	118.00	191.00	215.00	182.00	1
ps_sistolica	140.00	140.00	130.0	120.00	146.00	180.00	190.00	118.00	116.00	178.0	...	136.00	140.00	144.00	161.00	186.00	140.00	1
cintura	40.00	37.00	37.0	40.00	36.00	41.00	44.00	33.00	34.00	46.0	...	33.00	34.00	28.00	46.00	46.00	42.00	
prop_cin_cad	0.95	0.79	0.9	0.82	0.84	0.89	0.92	0.73	0.87	0.9	...	0.85	0.87	0.78	0.94	1.05	1.08	
Xn	1.00	1.00	1.0	1.00	1.00	1.00	1.00	1.00	1.00	1.0	...	1.00	1.00	1.00	1.00	1.00	1.00	

9 rows × 273 columns

```
X = X.to_numpy()
X
```

```
array([[169.  , 104.  ,   2.9 , ...,  40.  ,   0.95,   1.  ],
       [179.  ,  85.  ,   3.4 , ...,  37.  ,   0.79,   1.  ],
       [213.  ,  83.  ,   4.5 , ...,  37.  ,   0.9 ,   1.  ],
       ...,
       [160.  , 196.  ,   4.8 , ...,  49.  ,   1.09,   1.  ],
       [219.  , 173.  ,   7.1 , ...,  41.  ,   0.82,   1.  ],
       [199.  ,  85.  ,   3.4 , ...,  38.  ,   0.95,   1.  ]])
```

```
Xt = Xt.to_numpy()
Xt
```

```
array([[169.  , 179.  , 213.  , ..., 160.  , 219.  , 199.  ],
       [104.  ,  85.  ,  83.  , ..., 196.  , 173.  ,  85.  ],
       [  2.9 ,   3.4 ,   4.5 , ...,   4.8 ,   7.1 ,   3.4 ],
       ...,
       [ 40.  ,  37.  ,  37.  , ...,  49.  ,  41.  ,  38.  ],
       [  0.95,   0.79,   0.9 , ...,   1.09,   0.82,   0.95],
       [  1.  ,   1.  ,   1.  , ...,   1.  ,   1.  ,   1.  ]])
```

```
A = Xt@X
print(A)
```

```
[[1.2196441e+07 6.3338180e+06 2.6720720e+05 2.7043180e+06 1.0094207e+07
  7.8609770e+06 2.1513320e+06 4.9914480e+04 5.6355000e+04]
 [6.3338180e+06 4.1731520e+06 1.4358750e+05 1.4837400e+06 5.4745860e+06
  4.1949200e+06 1.1609460e+06 2.6721420e+04 2.9988000e+04]
 [2.6720720e+05 1.4358750e+05 6.4791700e+03 5.9335200e+04 2.2671140e+05
  1.7238980e+05 4.7850900e+04 1.1006290e+03 1.2377000e+03]
 [2.7043180e+06 1.4837400e+06 5.9335200e+04 6.7273800e+05 2.2684630e+06
  1.8125780e+06 4.8944800e+05 1.1403330e+04 1.2816000e+04]
 [1.0094207e+07 5.4745860e+06 2.2671140e+05 2.2684630e+06 9.1154830e+06
  6.7395260e+06 1.8998920e+06 4.3172470e+04 4.8613000e+04]
 [7.8609770e+06 4.1949200e+06 1.7238980e+05 1.8125780e+06 6.7395260e+06
  5.3356950e+06 1.4373650e+06 3.3360480e+04 3.7683000e+04]
 [2.1513320e+06 1.1609460e+06 4.7850900e+04 4.8944800e+05 1.8998920e+06
  1.4373650e+06 4.0231900e+05 9.2156900e+03 1.0361000e+04]
 [4.9914480e+04 2.6721420e+04 1.1006290e+03 1.1403330e+04 4.3172470e+04
  3.3360480e+04 9.2156900e+03 2.1467790e+02 2.4123000e+02]
 [5.6355000e+04 2.9988000e+04 1.2377000e+03 1.2816000e+04 4.8613000e+04
  3.7683000e+04 1.0361000e+04 2.4123000e+02 2.7300000e+02]]
```

```
A = pd.DataFrame(A)
A
```

	0	1	2	3	4	5	6	7	8
0	12196441.00	6333818.00	267207.200	2704318.00	10094207.00	7860977.00	2151332.00	49914.4800	56355.00
1	6333818.00	4173152.00	143587.500	1483740.00	5474586.00	4194920.00	1160946.00	26721.4200	29988.00

```
A = A.to_numpy()
A
```

```
array([[1.2196441e+07, 6.3338180e+06, 2.6720720e+05, 2.7043180e+06,
        1.0094207e+07, 7.8609770e+06, 2.1513320e+06, 4.9914480e+04,
        5.6355000e+04],
       [6.3338180e+06, 4.1731520e+06, 1.4358750e+05, 1.4837400e+06,
        5.4745860e+06, 4.1949200e+06, 1.1609460e+06, 2.6721420e+04,
        2.9988000e+04],
       [2.6720720e+05, 1.4358750e+05, 6.4791700e+03, 5.9335200e+04,
        2.2671140e+05, 1.7238980e+05, 4.7850900e+04, 1.1006290e+03,
        1.2377000e+03],
       [2.7043180e+06, 1.4837400e+06, 5.9335200e+04, 6.7273800e+05,
        2.2684630e+06, 1.8125780e+06, 4.8944800e+05, 1.1403330e+04,
        1.2816000e+04],
       [1.0094207e+07, 5.4745860e+06, 2.2671140e+05, 2.2684630e+06,
        9.1154830e+06, 6.7395260e+06, 1.8998920e+06, 4.3172470e+04,
        4.8613000e+04],
       [7.8609770e+06, 4.1949200e+06, 1.7238980e+05, 1.8125780e+06,
        6.7395260e+06, 5.3356950e+06, 1.4373650e+06, 3.3360480e+04,
        3.7683000e+04],
       [2.1513320e+06, 1.1609460e+06, 4.7850900e+04, 4.8944800e+05,
        1.8998920e+06, 1.4373650e+06, 4.0231900e+05, 9.2156900e+03,
        1.0361000e+04],
       [4.9914480e+04, 2.6721420e+04, 1.1006290e+03, 1.1403330e+04,
        4.3172470e+04, 3.3360480e+04, 9.2156900e+03, 2.1467790e+02,
        2.4123000e+02],
       [5.6355000e+04, 2.9988000e+04, 1.2377000e+03, 1.2816000e+04,
        4.8613000e+04, 3.7683000e+04, 1.0361000e+04, 2.4123000e+02,
        2.7300000e+02]])
```

```
Ainversa = np.linalg.inv(A)
Ainversa
```

```
array([[ 2.73868442e-06, -2.23419931e-08, -3.49228307e-05,
        -9.69025896e-07,  3.56421918e-07, -9.58889852e-07,
        -1.83392977e-06,  6.30974423e-05, -2.76329991e-04],
       [-2.23419931e-08,  1.38174839e-06, -7.24140443e-06,
        -1.37103447e-06, -3.06152046e-07,  6.81812754e-08,
        -1.52428103e-07, -5.00408284e-05,  4.51334337e-05],
       [-3.49228307e-05, -7.24140443e-06,  1.82413497e-03,
        -2.83050776e-06, -2.29117676e-05,  8.41352645e-06,
         4.49415183e-05, -3.30928607e-03,  4.00438880e-03],
       [-9.69025896e-07, -1.37103447e-06, -2.83050776e-06,
         2.19566191e-05,  4.33456771e-06, -5.54275016e-06,
        -2.15725022e-05, -3.79731260e-04,  4.80211379e-04],
       [ 3.56421918e-07, -3.06152046e-07, -2.29117676e-05,
         4.33456771e-06,  1.05249690e-05, -3.24881192e-07,
        -7.02558435e-05,  1.20323628e-03, -3.65724932e-04],
       [-9.58889852e-07,  6.81812754e-08,  8.41352645e-06,
        -5.54275016e-06, -3.24881192e-07,  9.96920286e-06,
        -4.60800475e-06,  1.30633441e-04, -8.46260469e-04],
       [-1.83392977e-06, -1.52428103e-07,  4.49415183e-05,
        -2.15725022e-05, -7.02558435e-05, -4.60800475e-06,
         6.36879407e-04, -1.40317690e-02,  2.57852766e-03],
       [ 6.30974423e-05, -5.00408284e-05, -3.30928607e-03,
        -3.79731260e-04,  1.20323628e-03,  1.30633441e-04,
        -1.40317690e-02,  1.07564987e+00, -6.24923579e-01],
       [-2.76329991e-04,  4.51334337e-05,  4.00438880e-03,
         4.80211379e-04, -3.65724932e-04, -8.46260469e-04,
         2.57852766e-03, -6.24923579e-01,  6.51323456e-01]])
```

```
Nueva = Ainversa@Xt
Nueva
```

```
array([[-3.40879719e-05, -2.62044800e-05,  3.62596697e-05, ...,
        -1.27742965e-04, -3.17659007e-05,  2.30436085e-08],
       [ 3.85455870e-05, -3.38980087e-07, -1.26496428e-05, ...,
         7.72192356e-05,  5.03142495e-05, -6.02386335e-05],
       [-1.12812646e-03, -6.25647131e-04,  2.61079547e-04, ...,
        -6.22915032e-04,  3.35309786e-03, -1.76939776e-03],
       ...,
       [ 2.52423405e-03,  9.58302863e-04,  9.72331290e-04, ...,
        -2.05290264e-03,  1.56635513e-03, -9.47428183e-04],
       [ 2.56291149e-02, -7.70296616e-02,  1.17397400e-02, ...,
         1.65430146e-01, -1.07691930e-01,  5.73465538e-02],
```

```
       [-3.23988224e-02,  5.24530828e-02, -4.38040515e-03, ...,
        -1.20332514e-01,  9.05467103e-02, -2.54436768e-02]])
```

```
Y = Y.to_numpy()
Y
```

```
array([[0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
       [1],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
```

```
ab = Nueva@Y
ab
```

```
array([[0.00047391011039231564],
       [0.004326064840941465],
       [0.010772014850325277],
       [0.000433571324178371],
       [-0.00045627172354809737],
       [0.0007264512732575491],
       [0.003952395309539705],
       [0.19105615281791233],
       [-0.8152372863441626]], dtype=object)
```

```
ab= pd.DataFrame(ab)
ab
```

	0
0	0.000474
1	0.004326
2	0.010772
3	0.000434
4	-0.000456
5	0.000726
6	0.003952
7	0.191056
8	-0.815237

```
print("La fórmula del problema es: y =", ab.loc[8,0],"+",ab.loc[7,0],"x","+",ab.loc[6,0],"x","+",ab.loc[5,0],"x","+",ab.loc[4,0],"x","+",a
```

```
La fórmula del problema es: y = -0.8152372863441626 + 0.19105615281791233 x + 0.003952395309539705 x + 0.0007264512732575491 x + -0.0004
```

```
print("Predicciones")
a  = ab.loc[8,0] + ab.loc[7,0]*test['prop_cin_cad'] + ab.loc[6,0]*test['cintura'] + ab.loc[5,0]*test['ps_sistolica'] + ab.loc[4,0]*test['p
```

```
Predicciones
```

```
dfPredicciones = pd.DataFrame(a)
```

```
dfY = pd.DataFrame(y_test)
```

```
dfY
dfY = dfY.rename(columns={ 0 :'Casos de Diabetes'})
```

```
Comparativa = pd.concat([dfY, dfPredicciones], axis =1 )
```

```
Comparativa
```

	diabetes	0
10	0	-0.486479
378	0	-0.468800
17	0	-0.515969
122	0	-0.483663
205	0	-0.434135
...	...	...
185	0	-0.484122
261	0	-0.431762
108	0	-0.511853
21	0	-0.527060
319	0	-0.463043

117 rows × 2 columns

```
Comparativa.columns
```

```
Index(['diabetes', 0], dtype='object')
```

```
Comparativa = Comparativa.rename(columns={0:'Predicción'})
Comparativa.columns
```

```
Index(['diabetes', 'Predicción'], dtype='object')
```

```
Comparativa
```

	diabetes	Predicción
10	0	-0.486479
378	0	-0.468800
17	0	-0.515969
122	0	-0.483663
205	0	-0.434135
...	...	...
185	0	-0.484122
261	0	-0.431762
108	0	-0.511853
21	0	-0.527060
319	0	-0.463043

117 rows × 2 columns

```
Comparativa['Predicción'].max()
```

```
-0.37682111871614155
```

```
Comparativa['Predicción'].min()
```

```
-0.5449043640453016
```

```
Comparativa.loc[Comparativa['Predicción'] > 0.268, 'Predicciones'] = 1
Comparativa.loc[Comparativa['Predicción'] < 0.268, 'Predicciones'] = 0
```

```
Comparativa
```

	diabetes	Predicción	Predicciones
10	0	-0.486479	0.0
378	0	-0.468800	0.0
17	0	-0.515969	0.0
122	0	-0.483663	0.0
205	0	-0.434135	0.0
...	...	...	...
185	0	-0.484122	0.0
261	0	-0.431762	0.0
108	0	-0.511853	0.0
21	0	-0.527060	0.0
319	0	-0.463043	0.0

117 rows × 3 columns

```
freq = Comparativa['diabetes'].value_counts()
freq
```

```
0    102
1     15
Name: diabetes, dtype: int64
```

```
freq = Comparativa['Predicciones'].value_counts()
freq
```

```
0.0    117
Name: Predicciones, dtype: int64
```

```
Comparativa.drop(['Predicción'], axis = 1)
```

	diabetes	Predicciones
10	0	0.0
378	0	0.0
17	0	0.0
122	0	0.0
205	0	0.0
...	...	...
185	0	0.0
261	0	0.0
108	0	0.0
21	0	0.0
319	0	0.0

117 rows × 2 columns

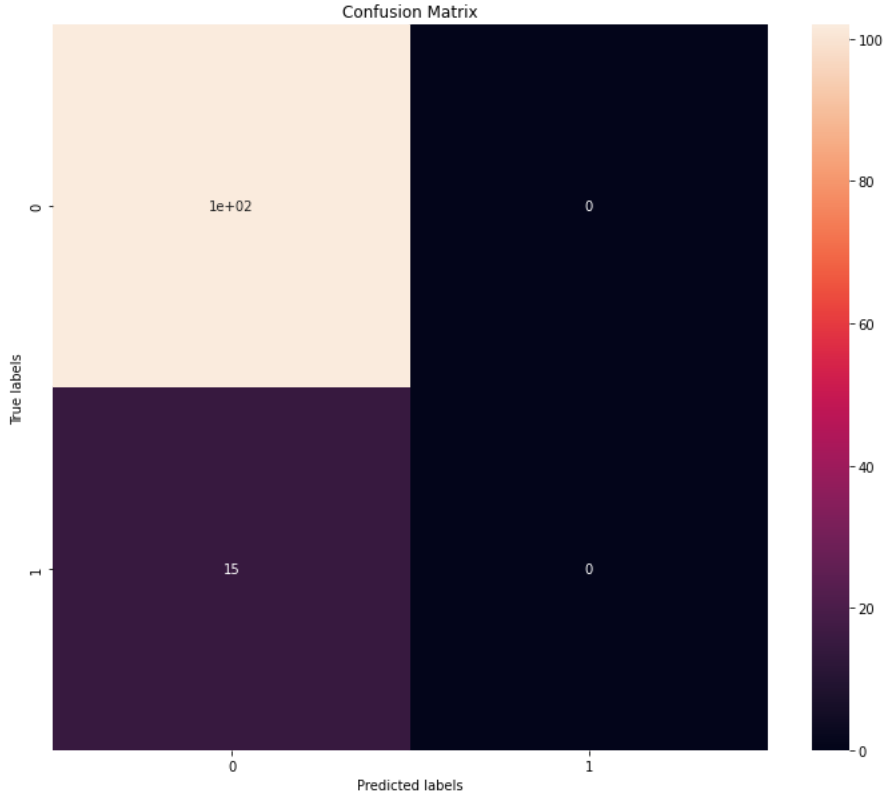
```
from sklearn.metrics import confusion_matrix

c = confusion_matrix(Comparativa['diabetes'].tolist(),Comparativa['Predicciones'].tolist())
np.flip(c.T)
```

```
array([[  0,   0],
       [ 15, 102]])
```

```
# Accuracy
from sklearn.metrics import accuracy_score
print(accuracy_score(Comparativa['diabetes'].tolist(),Comparativa['Predicciones'].tolist()))
```

```
0.8717948717948718
```

```
ax= plt.subplot()
sns.heatmap(c, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```



Análisis discriminante

```
from sklearn import preprocessing
df_discriminante_diabetes = df_discriminante['diabetes']
df_discriminante = df_discriminante.drop(['genero','diabetes'], axis = 1)
z_scaler = preprocessing.StandardScaler()
df_stan1 = pd.DataFrame(z_scaler.fit_transform(df_discriminante))
df_stan1 = df_stan1.set_axis(['colesterol', 'glucosa', 'hdl_chol', 'prop_col_hdl', 'edad', 'altura','peso','IMC','ps_sistolica','ps_diasto

X = df_stan1[['colesterol', 'glucosa', 'hdl_chol', 'prop_col_hdl','edad', 'altura', 'peso', 'IMC', 'ps_sistolica','ps_diastolica', 'cintur
y = df_discriminante_diabetes
model = LinearDiscriminantAnalysis()
model.fit(X, y)
cv = RepeatedStratifiedKFold(n_splits=12, n_repeats=5, random_state=40)
scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
print(np.mean(scores))
new = [199,77,49,3.9,19,61,119,22.5,118,70, 32, 38, 0.84]
model.predict([new])
```

```
0.9215909090909091
/usr/local/lib/python3.7/dist-packages/sklearn/base.py:451: UserWarning: X does not have valid feature names, but LinearDiscriminantAnal
  "X does not have valid feature names, but"
array(['Diabetes'], dtype='<U11')
```

```

from sklearn.model_selection import cross_val_predict
y_pred = cross_val_predict(model, X, y, cv=13)
```

```

from sklearn.metrics import accuracy_score
accuracy_score(y, y_pred)
```

```
0.9230769230769231
```

```
cm
```

```
array([[ 37,  23],
       [  7, 323]])
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, y_pred)
import  seaborn as sns
ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```


Confusion Matrix
True labels
0
1
37
23
7
3.2e+02
300
250
200
150
100