```
from google.colab import drive

drive.mount("/content/gdrive")
!pwd  # show current path
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call d
/content/gdrive/My Drive/ClasesMachineLearning
```

```
%cd "/content/gdrive/MyDrive/ClasesMachineLearning"
!ls  # show current directory
```

```
/content/gdrive/MyDrive/ClasesMachineLearning
brain_stroke.csv        MR2.ipynb             Valhalla23.csv
ChallengeSemana2.ipynb  Semana3.ipynb         wine.data
iris.data               Ses03_Practice.ipynb  wine.names
```

```
import numpy as np
from random import randrange
import matplotlib.pyplot as plt
import math
import pandas as pd
import seaborn as sn
```

```
df = pd.read_csv('brain_stroke.csv') # Leer dataset
df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_
0	Male	67.0	0	1	Yes	Private	
1	Male	80.0	0	1	Yes	Private	
2	Female	49.0	0	0	Yes	Private	
3	Female	79.0	1	0	Yes	Self-employed	
4	Male	81.0	0	0	Yes	Private	

```
df['gender'] = df['gender'].map({'Male':0,'Female':1}) #Transformamos la información c
df['ever_married'] = df['ever_married'].map({'Yes':0,'No':1}) # Hacemos lo mismo con l
fSmoked = []
nSmoked = []
smokes = []
```

```
unknown = []

for i in df.values:
  if i[9] == "formerly smoked":
    fSmoked.append(1)
  else:
    fSmoked.append(0)
  if i[9] == "never smoked":
    nSmoked.append(1)
  else:
    nSmoked.append(0)
  if i[9] == "smokes":
    smokes.append(1)
  else:
    smokes.append(0)
  if i[9] == "Unknown":
    unknown.append(1)
  else:
    unknown.append(0)

df['fSmoked'] = fSmoked
df['nSmoked'] = nSmoked
df['smokes'] = smokes
df['sUnknown'] = unknown
#Transformamos las variables de fumador en variables dummy con 1 y 0 cada una
df.head()
```

	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_t
0	0	67.0	0	1	0	Private	Ur
1	0	80.0	0	1	0	Private	R
2	1	49.0	0	0	0	Private	Ur
3	1	79.0	1	0	0	Self-employed	R
4	0	81.0	0	0	0	Private	Ur

```
sn.set(rc = {'figure.figsize':(25,16)})
sn.heatmap(df.corr(), annot=True, cmap= 'YlGnBu') # Revisar correlación de variables p
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feba58b2a10>
```

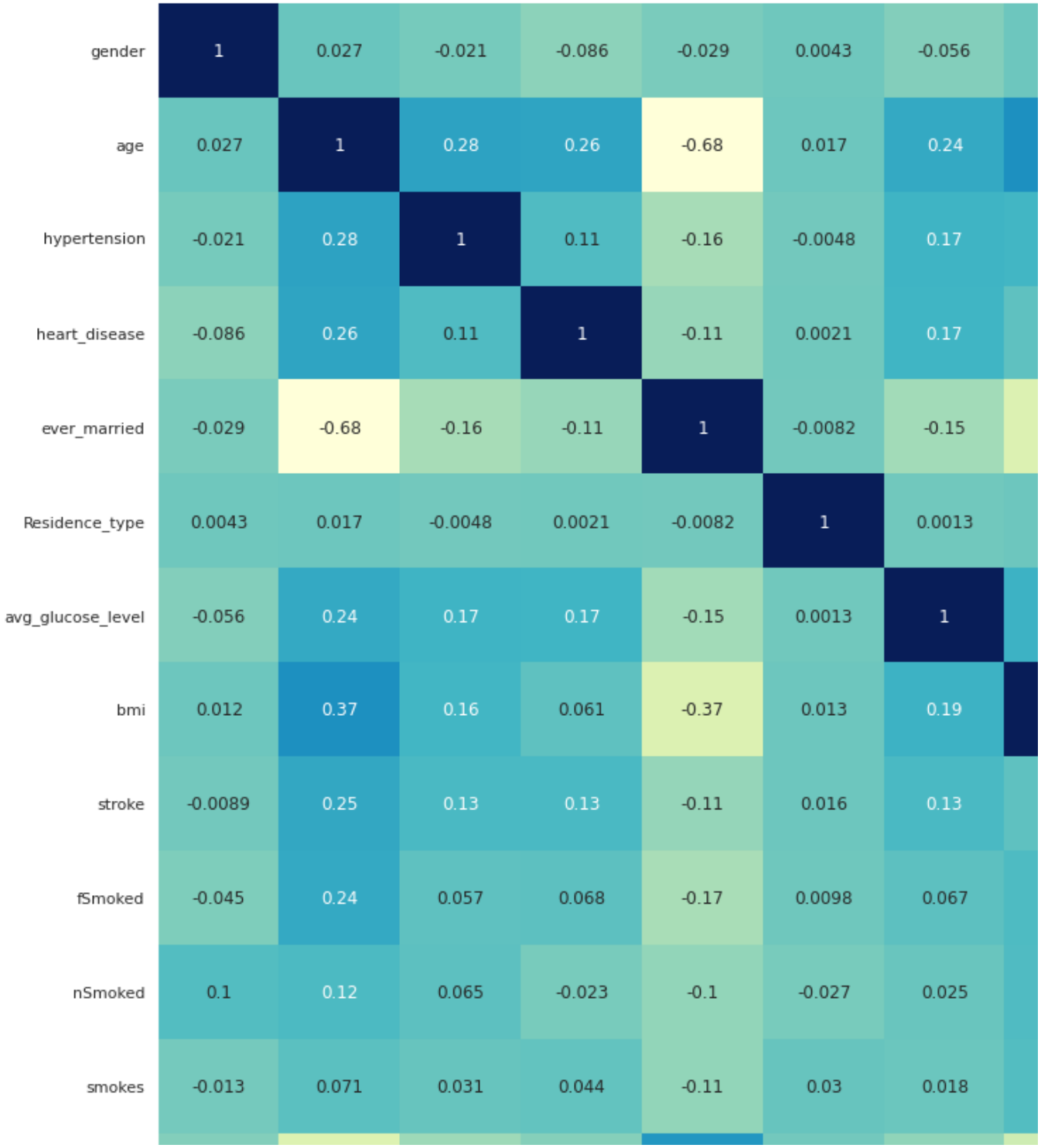

```
df = df.drop(["work_type", "smoking_status"], axis=1)
df['Residence_type'] = df['Residence_type'].map({'Rural':0,'Urban':1})
df.head() # Eliminamos algunas columnas y cambiamos el tipo de residencia a 1 y 0
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_gl
0	0	67.0	0	1	0	1	
1	0	80.0	0	1	0	0	
2	1	49.0	0	0	0	1	
3	1	79.0	1	0	0	0	
4	0	81.0	0	0	0	1	

```
sn.set(rc = {'figure.figsize':(25,16)})
```

```
sn.heatmap(df.corr(), annot=True, cmap= 'YlGnBu') #Visualizamos la correlación de las
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feba5923c50>
```

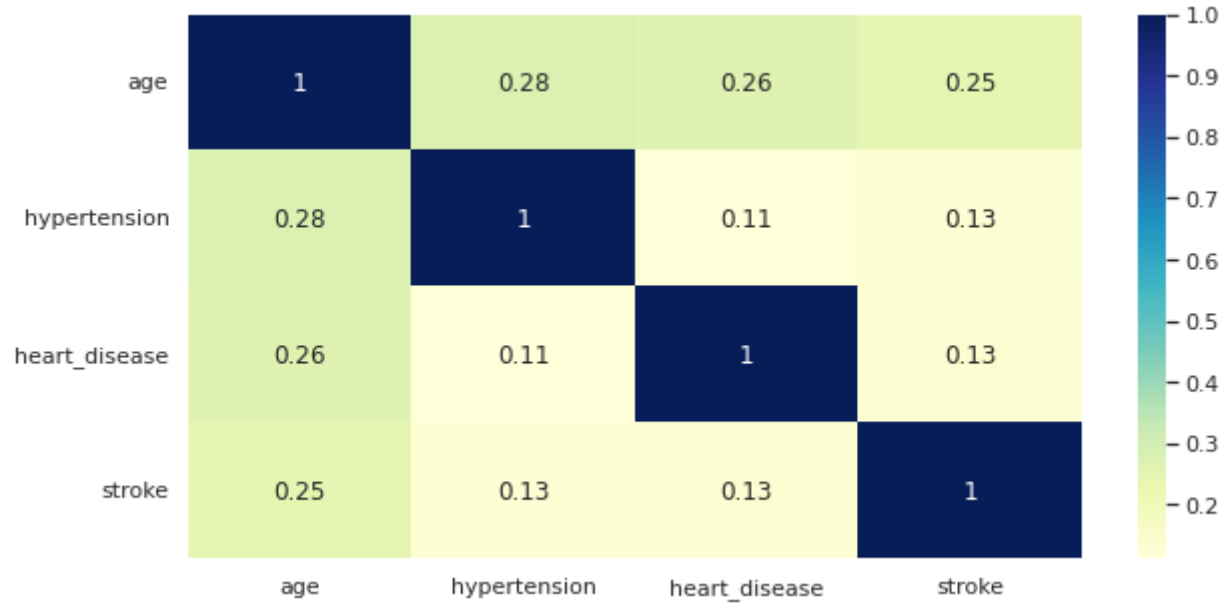



```
df = df.drop(["gender", "Residence_type","avg_glucose_level","nSmoked","smokes","sUnkr
df.head() #Eliminamos todas las variables que no influyen en nuestro resultado
```

	age	hypertension	heart_disease	stroke
0	67.0	0	1	1
1	80.0	0	1	1

```
sn.set(rc = {'figure.figsize':(10,5)})
sn.heatmap(df.corr(), annot=True, cmap= 'YlGnBu') # Visualizamos solo las variables ut
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feba553b7d0>
```




```
df_x = df.drop(["stroke"],axis=1).values
df_y = df["stroke"].values
print(len(df_x)) #Separamos nuestra información de entrada y salida de nuestro modelo
```

```
4981
```

```
train_y = []
train_x = []
validate_y = []
validate_x = []
for i in range(0,4000):
  value = randrange(0,len(df_y))
  train_y.append(df_y[value])
  train_x.append(df_x[value])
  df_y = np.concatenate((df_y[:value],df_y[value+1:]))
  df_x = np.concatenate((df_x[:value],df_x[value+1:]))
validate_y = df_y
validate_x = df_x
#Aleatoriamente tomamos 4000 valores de nuestros datos para entrenar y dejamos los otr
```

```
h   = lambda x,theta: theta[0]+theta[1]*x[0]+theta[2]*x[1]+theta[3]*x[2]
j_i = lambda x,y,theta: (y - h(x,theta))**2 #Creamos nuestra función h y nuestra funci
```

```
theta = [1,1,1,1,1] #Cambia dependiendo del orden del modelo (1 theta para cada dimens
alpha = 0.01
n = len(train_y)
print(theta) #Realizamos 1000 iteraciones con un alpha de 0.01
for idx in range(1000):
  acumDelta0 = []
  acumDelta1 = []
  acumDelta2 = []
  acumDelta3 = []
  for x_i, y_i in zip(train_x,train_y):
    acumDelta0.append(h(x_i,theta)-y_i)
    acumDelta1.append((h(x_i,theta)-y_i)*x_i[0])
    acumDelta2.append((h(x_i,theta)-y_i)*x_i[1])
    acumDelta3.append((h(x_i,theta)-y_i)*x_i[2])
  sJt0 = sum(acumDelta0)
  sJt1 = sum(acumDelta1)
  sJt1 = sum(acumDelta2)
  sJt1 = sum(acumDelta3)
  theta[0] = theta[0] - (alpha/n)*sJt0
  theta[1] = theta[1] - ((alpha/n)*sJt1)
  theta[2] = theta[2] - ((alpha/n)*sJt1)
  theta[3] = theta[3] - ((alpha/n)*sJt1)
print(theta)
```

```
[1, 1, 1, 1, 1]
[-0.5733452394969707, 0.011214290464904483, 0.011214290464904483, 0.011214290464
```

```
n_train = len(train_y)
n_validate = len(validate_y)

#Validación
acumDelta = []
for x_i, y_i in zip(validate_x,validate_y):
    acumDelta.append(j_i(x_i,y_i,theta))

sDelta = sum(acumDelta)
j_validate = 1/(2*n_validate)*sDelta

print(j_validate)

#Trainingg
acumDelta = []
for x_i, y_i in zip(train_x,train_y):
    acumDelta.append(j_i(x_i,y_i,theta))

sDelta = sum(acumDelta)
j_train = 1/(2*n_train)*sDelta

print(j_train)
```

```
print(theta)
#Obtenemos el error el cual es muy pequeño
```

```
0.05256122803219075
0.05138047805995558
[-0.5733452394969707, 0.011214290464904483, 0.011214290464904483, 0.011214290464
```

```
print(df)
```

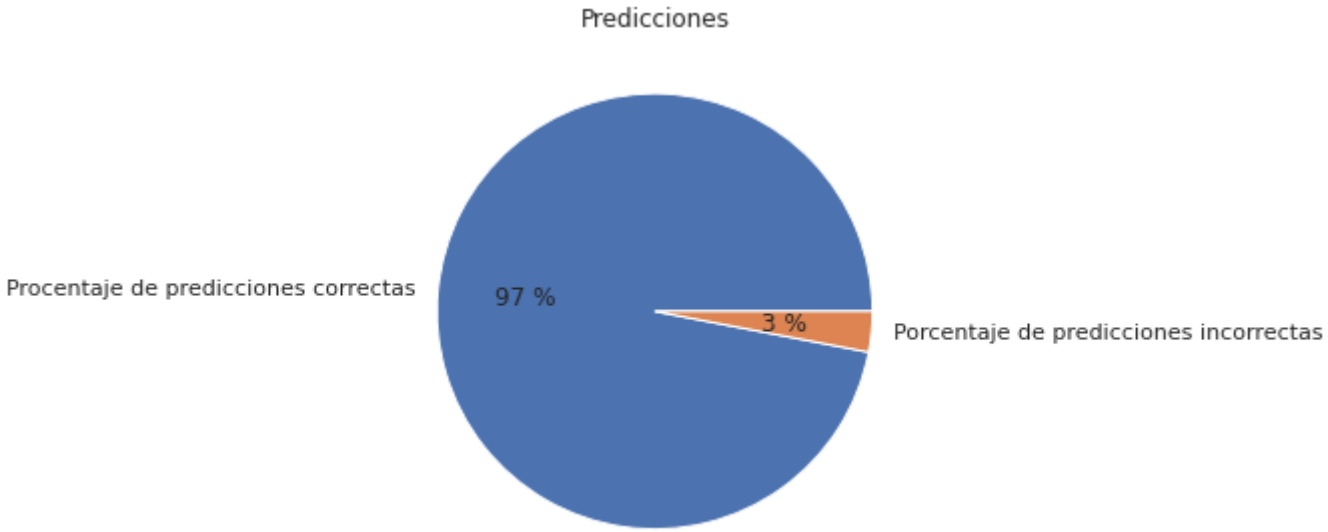
```
        age  hypertension  heart_disease  stroke
0     67.0             0              1       1
1     80.0             0              1       1
2     49.0             0              0       1
3     79.0             1              0       1
4     81.0             0              0       1
...    ...           ...            ...     ...
4976  41.0             0              0       0
4977  40.0             0              0       0
4978  45.0             1              0       0
4979  40.0             0              0       0
4980  80.0             1              0       0

[4981 rows x 4 columns]
```

```
def predict(x,expected):
  res = h(x,theta)
  exp = 0
  if res<0.5:
    exp =  0
  else:
    exp = 1
  if exp == expected:
    return 1
  else:
    return 0

#Creamos una función que de acuerdo al resultado de nuestro modelo pueda predecir un r
#Y retornar si el modelo predijo bien o no, si predice bien retorna 1 y si predice mal

#Para redondear, ya que queremos resultados de 1 y 0, si el valor es menor a 0.5, asum



valid_y = validate_y
valid_x = validate_x
good = 0
bad = 0
l = 981
for i in range(0,100):
  n = randrange(0,l)
  if predict(valid_x[n],valid_y[n]) == 1:
    good+=1
```

```
    else:
      bad+=1
    valid_x = np.concatenate((valid_x[:n],valid_x[n+1:]))
    valid_y = np.concatenate((valid_y[:n],valid_y[n+1:]))
    l-=1
print(good,bad)
# Aleatoriamente seleccionamos 100 valores de nuestra muestra de validación y los mete
# veces predice correctamente y cuantas se equivoca
```

```
97 3
```

```
efectividad = [good, bad]
titulo = ["Procentaje de predicciones correctas", "Porcentaje de predicciones incorrec
plt.title("Predicciones")
plt.pie(efectividad, labels=titulo, autopct="%0.0f %%")
plt.show()
#Graficamos el porcentaje de aciertos de nuestro modelo y podemos visualizar que es ur
```



Productos de pago de Colab - Cancelar contratos

✓ 0 s completado a las 15:39