

# PROYECTO FINAL

Matias Ibarra – Año 2023

Curso Programación de Sistemas Informáticos



# PROBABILIDAD DE INGRESO A LA UNIVERSIDAD

---

*Por Matías Ibarra, Julio de 2023*

## Descripción general del proyecto:

El siguiente trabajo tiene, como objetivo principal, aplicar Python (y sus librerías) para hacer una introducción en la exploración de los datos y las técnicas de machine learning. Antes que nada, remarcar, que es una introducción en el tema, por lo que puede que haya varias mejoras por hacer. El objetivo principal, entonces, es aprender a través de la puesta en práctica de los conocimientos adquiridos hasta el momento en que se realizó el proyecto. Por supuesto que, para un futuro, cualquier sugerencia podrá ser tomada en cuenta.

## Descripción de los datos:

El punto de partida del proyecto es un dataset que describe la probabilidad que tienen los alumnos de la India de ingresar a determinadas universidades, según las calificaciones obtenidas en un conjunto de evaluaciones.

Enlace: <https://www.kaggle.com/datasets/ranitsarkar01/jamboree-linear-regression-dataset>

## Descripción de las columnas del dataset:

Campo	Tipo de Campo	Descripción
SERIALNUMBER	Int	Identificador de cada alumno registrado
GRE	Int	Examen que mide el razonamiento verbal, razonamiento cuantitativo, la escritura analítica y habilidades de pensamiento crítico
TEOFL	Int	Prueba estandarizada de dominio del idioma inglés
UNIVERSITYRATING	Float	Ranking universitario
SOP	Float	Declaración de Propósito
LOR	Float	Carta de recomendación
CGPA	Float	El promedio acumulativo de calificaciones
RESEARCH	Boolean	Si el estudiante tiene experiencia en investigación o no
CHANCEOFADMIT	Float	La probabilidad de ser admitido en la universidad

## Objetivo:

Se solicita facilitar un modelo que permita a los futuros alumnos saber que chances tienen de ingresar a la universidad según las calificaciones que obtuvieron en sus evaluaciones, esto puede aplicarse, por ejemplo, a una página web que visualice dicha predicción.

## LIMPIEZA DE DATOS

La primera parte del proyecto consiste en limpiar los datos para el correcto desarrollo del análisis posterior. En este caso, el dataset original estaba ya limpio, pero, como se indico en la descripción del proyecto, el objetivo de este es aprender, por lo que se introducen datos erróneos manualmente para su posterior depuración.

```
#Importamos las librerías principales.
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#Cargamos el dataset jamboree
ruta=
r'C:/Users/matia/OneDrive/Escritorio/Curso_The_Corner/jamboree/jamboree_dataset.csv'
df = pd.read_csv(ruta)
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

El primer paso, antes de manipular los datos para que ya no estén limpios, es cambiar el nombre de las columnas para que sea mas sencillo declararlas cuando sea necesario:

```
#Cambiamos el nombre de las columnas
df = df.rename(columns={'Serial No.': 'SERIALNUMBER', 'GRE Score': 'GRE', 'TOEFL Score': 'TOEFL', 'University Rating': 'UNIVERSITYRATING', 'Research': 'RESEARCH', 'Chance of Admit': 'CHANCEOFADMIT'})
df.head()
```

	SERIALNUMBER	GRE	TOEFL	UNIVERSITYRATING	SOP	LOR	CGPA	RESEARCH	CHANCEOFADMIT
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

Luego, se duplican columnas para que exista mas de una entrada con el mismo SERIALNUMBER, esto significaría que hay alumnos que se ingresaron en la base de datos mas de una vez, por lo que hay que eliminar esas repeticiones.

Para duplicar filas se utiliza la función pandas.DataFrame.copy() y pandas.DataFrame.append() que copia los datos y los agrega al dataframe, respectivamente. Luego, con pandas.DataFrame.reset\_index() se reestablecen los índices:

```
#Duplicamos filas:
duplicated_row = df.iloc[498].copy()
df = df.append(duplicated_row)
df.reset_index(drop=True, inplace=True)
duplicated_row = df.iloc[499].copy()
df = df.append(duplicated_row)
df.reset_index(drop=True, inplace=True)
duplicated_row = df.iloc[500].copy()
df = df.append(duplicated_row)
df.reset_index(drop=True, inplace=True)
#vemos si hay filas duplicadas:
duplicate_rows_df = df[df.duplicated()]
print("número de filas duplicadas: ", duplicate_rows_df.shape)
```

número de filas duplicadas: (3, 9)

```
#Vemos el índice de las filas duplicadas (500, 501 y 502):
print(df.duplicated())
```

```
0      False
1      False
2      False
...
499    False
500     True
501     True
502     True
Length: 503, dtype: bool
```

```
#Eliminamos las filas duplicadas:
df.drop(502, inplace=True)
df.drop(501, inplace=True)
df.drop(500, inplace=True)
#Comprobamos:
duplicate_rows_df = df[df.duplicated()]
print("número de filas duplicadas: ", duplicate_rows_df.shape)
```

número de filas duplicadas: (0, 9)

Hasta aquí, se vio como buscar y eliminar valores que estén duplicados, esto puede ocurrir por ejemplo porque un dato se ingresó dos veces por error. Puede ocurrir con frecuencia y es una de las tareas principales a la hora de hacer una limpieza de nuestros datos. A continuación, se ven otros dos sucesos que pueden ocurrir con frecuencia, los datos nulos (NaN) y los valores atípicos. Nuevamente, se modifican los datos manualmente para obtenerlos y poder hacer la limpieza de estos:

```
#Alteramos valores de celdas.
df.at[4, "TOEFL"] = 160      #valor atípico
df.at[5, "CGPA"] = None     #valor nulo
df.at[7, "GRE"] = None      #valor nulo
#Veamos ahora que count() nos dirá los valores faltantes (GRE Y CGPA tienen un
valor menos que el resto)
df.count()
```

```
SERIALNUMBER 503      SOP 503
GRE 502             LOR 503
TOEFL 503           CGPA 502
UNIVERSITYRATING 503 RESEARCH 503
CHANCEOFADMIT 503
dtype: int64
```

Llega el momento de buscar si hay datos atípicos (ya se sabe que al menos hay uno, porque se agregó intencionalmente). Para esta tarea se grafica un diagrama de caja, del que no se va a entrar demasiado en detalle pero si a continuación se da un resumen de cómo funciona:

*El diagrama de caja y bigotes, también llamado diagrama de caja o boxplot, es un gráfico que representa un conjunto de datos estadísticos de manera visual utilizando los cuartiles.*

*La principal característica del diagrama de caja y bigotes es que permite visualizar rápidamente la dispersión de una serie de datos, ya que indica los cuartiles, la mediana, los valores extremos y los valores atípicos de los datos.*

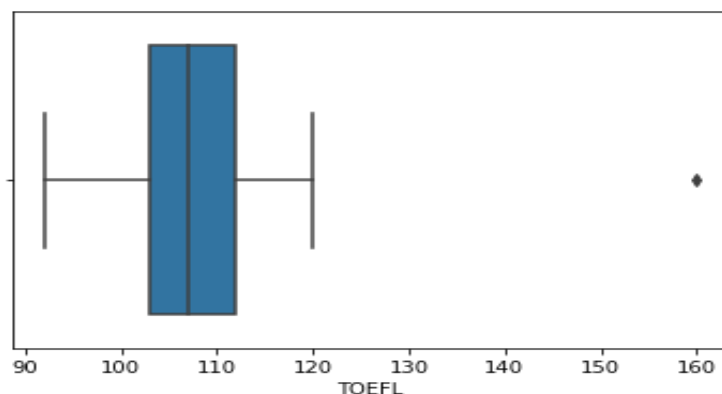
*Así pues, este tipo de diagrama está formado por una caja rectangular y unas líneas (o bigotes) de los cuales destacan los siguientes valores:*

- Los límites de la caja indican el primer y el tercer cuartil ( $Q_1$  y  $Q_3$ ). Y la línea vertical dentro de la caja es la mediana (equivalente al segundo cuartil  $Q_2$ ).
- Los límites de los bigotes (o brazos) son los valores extremos, es decir, el valor mínimo y el valor máximo de la serie de datos.
- Los puntos fuera de los bigotes son los valores atípicos (outliers), o dicho con otras palabras, datos que probablemente se han medido mal y por tanto no deberían tenerse en cuenta en el estudio estadístico.

Fuente: <https://www.probabilidadyestadistica.net/diagrama-de-caja-y-bigotes-boxplot/>

```
#Buscamos visualmente valores atípicos, encontramos el valor atípico en TOEFL.
sns.boxplot(x=df['TOEFL'])
```

```
<AxesSubplot: xlabel='TOEFL'>
```



Visualmente, se ubica el valor atípico para la columna TOEFL, es 160. Ahora se determina en que fila está ubicado este valor para eliminarla de la base de datos. Para esto se utiliza la formula:

$$q < Q1 - (1,5).IQR$$
$$q > Q3 + (1,5).IQR$$

Donde Q1 es el primer cuartil, Q3 el tercero, IQR el rango intercuartil y q serán los valores que cumplan la condición definida arriba, es decir, los outliers o valores atípicos.

```
#Definimos los cuartiles:
df['TOEFL'].quantile([0.25, 0.5, 0.75]) #q1, q2 y q3

0.25    103.0
0.50    107.0
0.75    112.0

#Calcular los límites para determinar los valores atípicos
q1 = 103
q3 = 112
iqr = q3 - q1
lower_bound = q1 - 1.5 * iqr
upper_bound = q3 + 1.5 * iqr
#Encontrar el número de fila del valor atípico
outliers = df[(df['TOEFL'] < lower_bound) | (df['TOEFL'] > upper_bound)]
indices_outliers = outliers.index.tolist()
#Imprimir el número de fila del valor atípico
for index in indices_outliers:
    print("Número de fila del valor atípico: ", index)
```

Número de fila del valor atípico: 4

```
#Optamos por eliminar la fila completa:
df.drop(4, inplace=True)
```

Queda una ultima tarea en lo que respecta a la limpieza de datos, eliminar las filas que contienen valores nulos, según las circunstancias, se puede también reemplazar los valores nulos por el valor de la media de todos los datos de la columna, por ejemplo. Esto no es recomendable en este caso porque estaríamos alterando el resultado final de los cálculos que realizaremos más adelante.

```
#Eliminamos las filas con valores nulos, recordemos que estaban en GRE y CGPA
#Buscamos su ubicación y vemos que están en las filas 5 y 7
print(df.loc[(df['GRE'].isnull() == True), 'GRE'])
print(df.loc[(df['CGPA'].isnull() == True), 'CGPA'])

7    NaN
Name: GRE, dtype: float64
5    NaN
Name: CGPA, dtype: float64
```

```
#Las eliminamos:
df.drop(5, inplace=True)
df.drop(7, inplace=True)
#Comprobamos, ya no hay valores nulos:
df.count()
```

```
SERIALNUMBER 497      GRE 497
TOEFL 497      UNIVERSITYRATING 497
SOP 497      LOR 497
CGPA 497      RESEARCH 497
CHANCEOFADMIT 497
dtype: int64
```

Se da por concluida la etapa de limpieza de datos, se observa que en este dataset no hay entradas repetidas, valores nulos ni valores atípicos. Se continua con el proyecto, de ahora en más, se trabajará con el dataset limpio, es por eso que conviene guardarlo con otro nombre para que, al abrirlo, los pasos previos de limpieza ya estén efectuados.

```
#Guardamos en un archivo csv nuevo
df.to_csv("C:/Users/matia/OneDrive/Escritorio/Curso_The_Corner/jamboree/jamboree_datoslimpios_22junio.csv")
```

## CORRELACIONES

En esta parte del proyecto, el énfasis se hace sobre la relación existente entre las distintas variables con la chance de admisión final, ya que este es el objetivo buscado.

El primer paso es cargar la base de datos con la limpieza realizada en el paso anterior:

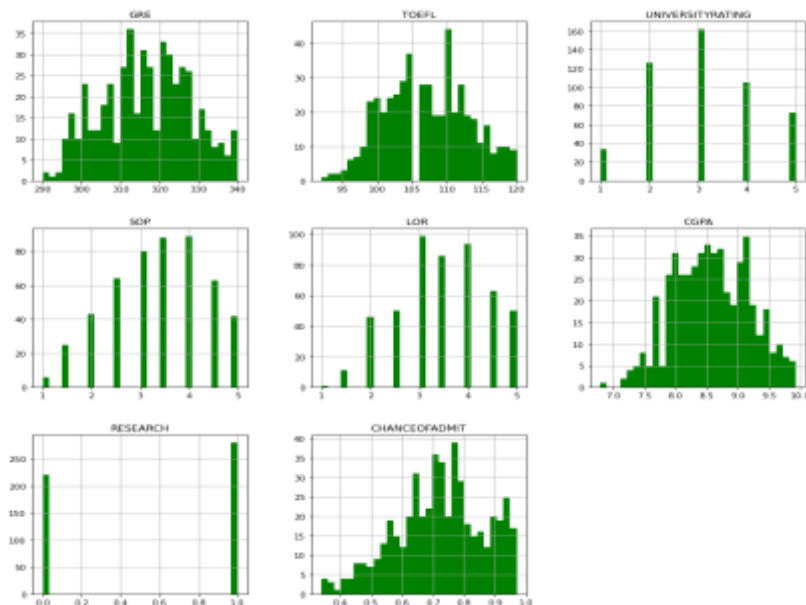
```
#Cargamos el dataset jamboree
ruta=
r'C:/Users/matia/OneDrive/Escritorio/Curso_The_Corner/jamboree/jamboree_datos1
imprios_22junio.csv'
df = pd.read_csv(ruta)
df.head()
```

	Unnamed: 0	SERIALNUMBER	GRE	TOEFL	UNIVERSITYRATING	SOP	LOR	CGPA	RESEARCH	CHANCEOFADMIT
0	0	1.0	337.0	118.0	4.0	4.5	4.5	9.65	1.0	0.92
1	1	2.0	324.0	107.0	4.0	4.0	4.5	8.87	1.0	0.76
2	2	3.0	316.0	104.0	3.0	3.0	3.5	8.00	1.0	0.72
3	3	4.0	322.0	110.0	3.0	3.5	2.5	8.67	1.0	0.80
4	6	7.0	321.0	109.0	3.0	3.0	4.0	8.20	1.0	0.75

```
#Eliminamos la primer columna:
df = df.drop(["Unnamed: 0"], axis=1)
```

Luego de esto, se ve como se distribuyen los valores en los distintos campos:

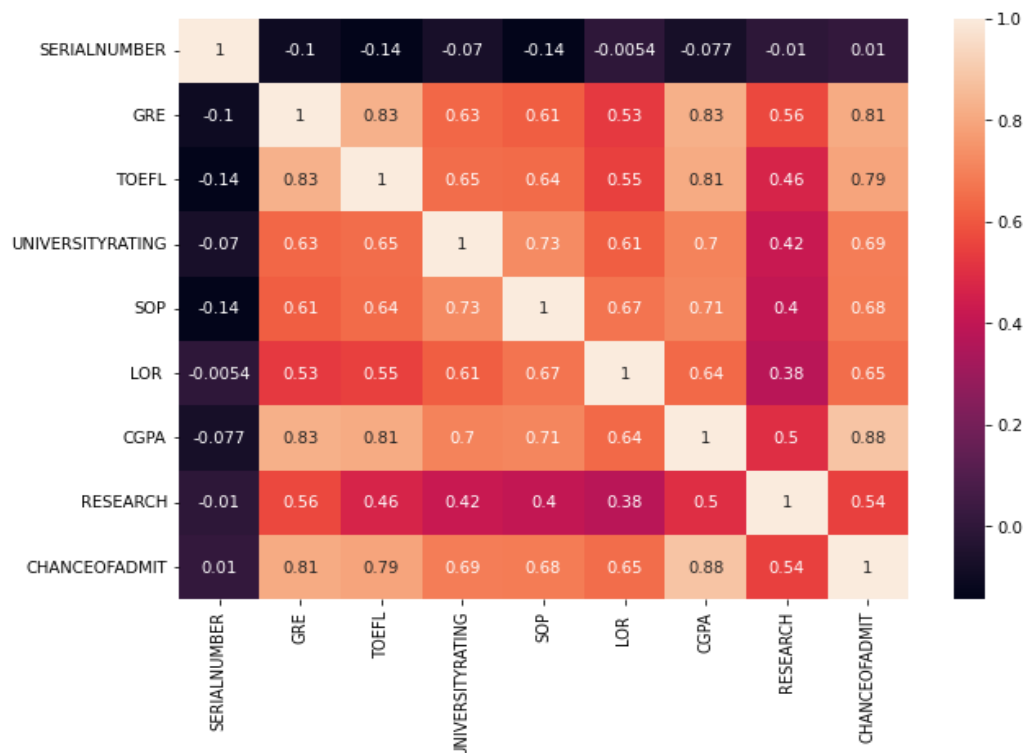
```
df_sinserialnumber = df.drop(['SERIALNUMBER'], axis=1)
df_sinserialnumber.hist(bins = 30, figsize=(15,15), color = 'green')
```





El paso siguiente es realizar un mapa de calor con las correlaciones existentes entre las distintas variables, la función `DataFrame.corr()` calcula la correlación por pares de columnas, por defecto con el método de Pearson, también podría utilizarse el método de Kendall o Spearman indicándolo de la siguiente manera `DataFrame.corr(method='kendall')`. Luego para graficar se puede utilizar `sns.heatmap()`:

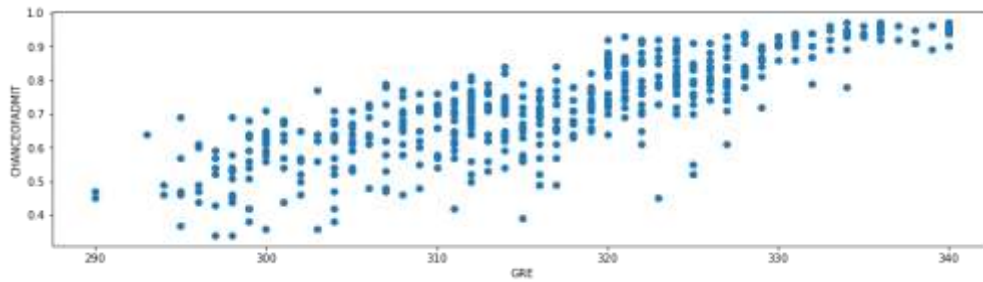
```
correlations = df.corr()
f, ax = plt.subplots(figsize = (10,7))
sns.heatmap(correlations, annot = True)
```



Gráficamente se puede apreciar la correlación existente entre las distintas variables, especialmente entre CHANCEOFADMIT y el resto, el mapa de calor permite distinguir aquellas correlaciones más significativas a través de colores mas claros dentro del recuadro. Por ejemplo, se observa que la chance de ser admitido tiene más correlación con la calificación obtenida en CGPA que en el resto.

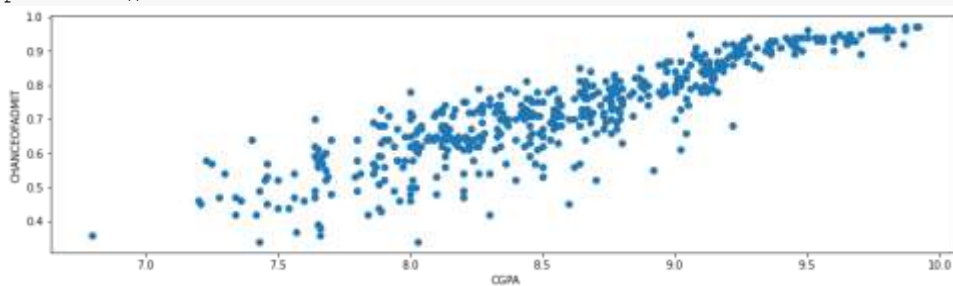
Luego, mediante la función `plt.scatter()` se ve la relación de cada variable con CHANCEOFADMIT:

```
plt.figure(figsize = (15,4))
plt.scatter(df['GRE'], df['CHANCEOFADMIT'])
plt.xlabel('GRE')
plt.ylabel('CHANCEOFADMIT')
plt.show()
```

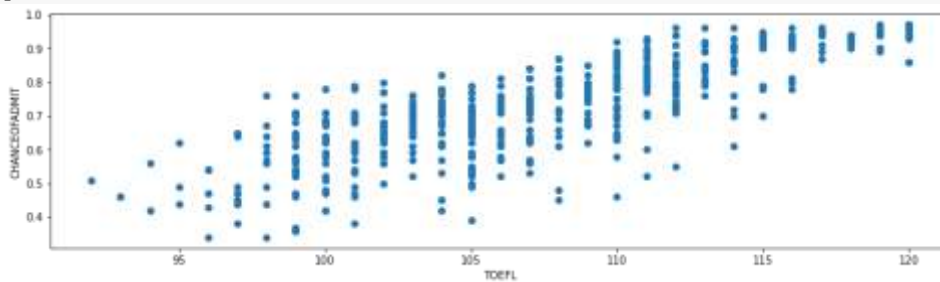


Se observa que, a mayor calificación, mayor es la chance de admisión. Lo mismo ocurre con el resto de las calificaciones:

```
plt.figure(figsize = (15,4))
plt.scatter(df['CGPA'], df['CHANCEOFADMIT'])
plt.xlabel('CGPA')
plt.ylabel('CHANCEOFADMIT')
plt.show()
```



```
plt.figure(figsize = (15,4))
plt.scatter(df['CGPA'], df['CHANCEOFADMIT'])
plt.xlabel('CGPA')
plt.ylabel('CHANCEOFADMIT')
plt.show()
```

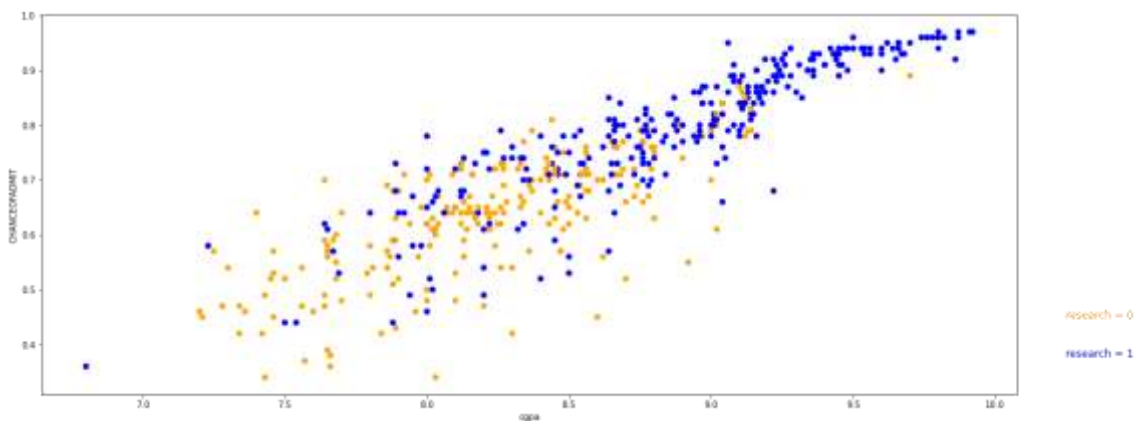


También se observa que, los estudiantes que tienen experiencia en investigación obtienen mejores calificaciones y son, en su mayoría, quienes tienen las chances más altas de admisión:

```

colors = []
for research in df['RESEARCH']:
    if research:
        colors.append('blue') # Valor True, color azul
    else:
        colors.append('orange') # Valor False, color naranja
plt.figure(figsize = (10,4))
plt.scatter(df['CGPA'], df['CHANCEOFADMIT'], c = colors)
plt.text(1.05, 0.1, 'research = 1', color='blue',
transform=plt.gca().transAxes, fontsize=12)
plt.text(1.05, 0.2, 'research = 0', color='orange',
transform=plt.gca().transAxes, fontsize=12)
plt.ylabel('CHANCEOFADMIT')
plt.xlabel('cgpa')
plt.show()

```

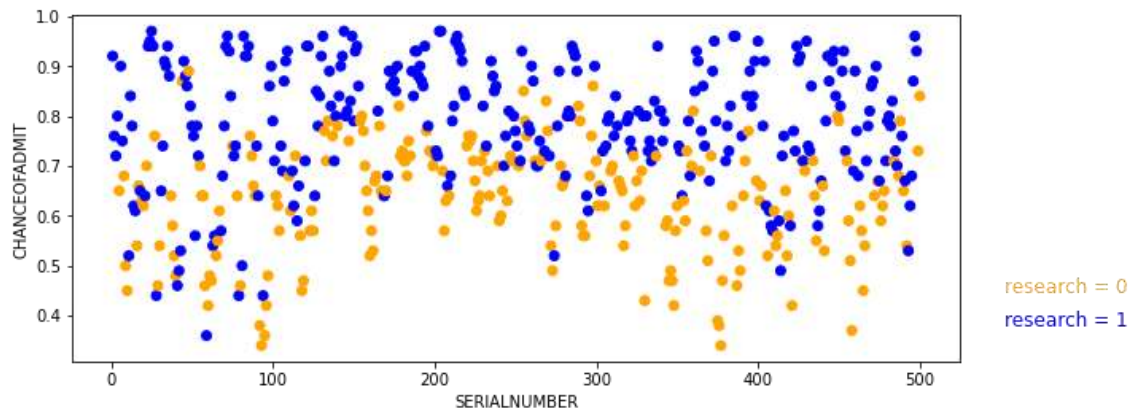


Los alumnos con experiencia en investigación son los puntos azules, mientras que, los de amarillo son los que no tienen experiencia. Se puede observar también como se distribuyen según el orden de registro y la chance con la que cuenta c/u:

```

colors = []
for research in df['RESEARCH']:
    if research:
        colors.append('blue') # Valor True, color azul
    else:
        colors.append('orange') # Valor False, color naranja
plt.figure(figsize = (10,4))
plt.scatter(df['SERIALNUMBER'], df['CHANCEOFADMIT'], c = colors)
plt.text(1.05, 0.1, 'research = 1', color='blue',
transform=plt.gca().transAxes, fontsize=12)
plt.text(1.05, 0.2, 'research = 0', color='orange',
transform=plt.gca().transAxes, fontsize=12)
plt.ylabel('CHANCEOFADMIT')
plt.xlabel('SERIALNUMBER')
plt.show()

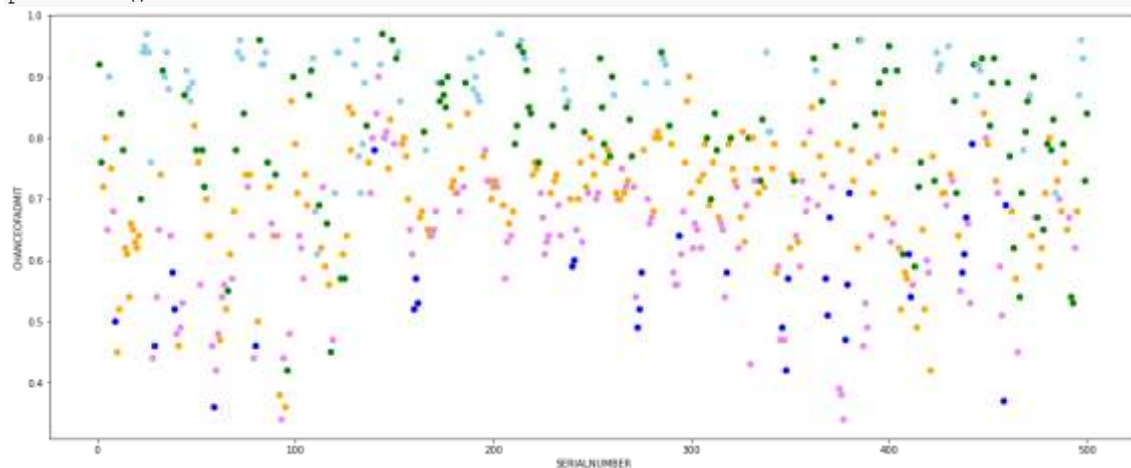
```



Son pocos los que superan el 80% de chance de admisión entre los que no cuentan con experiencia en investigación. En cambio, el resto, se ubican en su mayoría por encima del 70%.

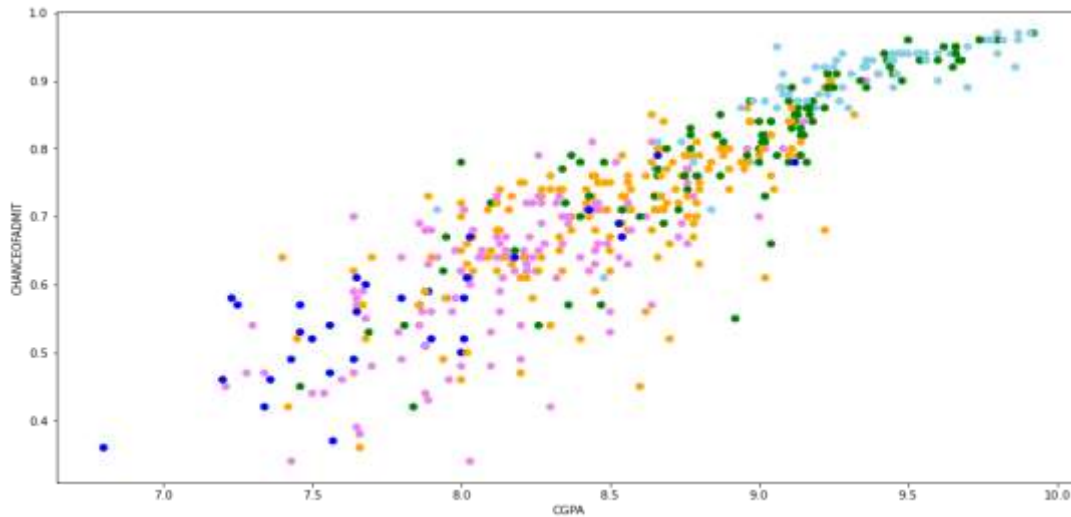
En cuanto a los resultados obtenidos y la jerarquía de la universidad a la que se desea ingresar, se observa que a mayor jerarquía, mayor es la calificación de los estudiantes:

```
colors = []
for universidad in df['UNIVERSITYRATING']:
    if universidad == 1:
        colors.append('blue')
    elif universidad == 2:
        colors.append('violet')
    elif universidad == 3:
        colors.append('orange')
    elif universidad == 4:
        colors.append('green')
    else:
        colors.append('skyblue')
plt.figure(figsize = (10,4))
plt.scatter(df['SERIALNUMBER'], df['CHANCEOFADMIT'], c = colors, cmap =
universidad)
plt.ylabel('CHANCEOFADMIT')
plt.xlabel('SERIALNUMBER')
plt.show()
```

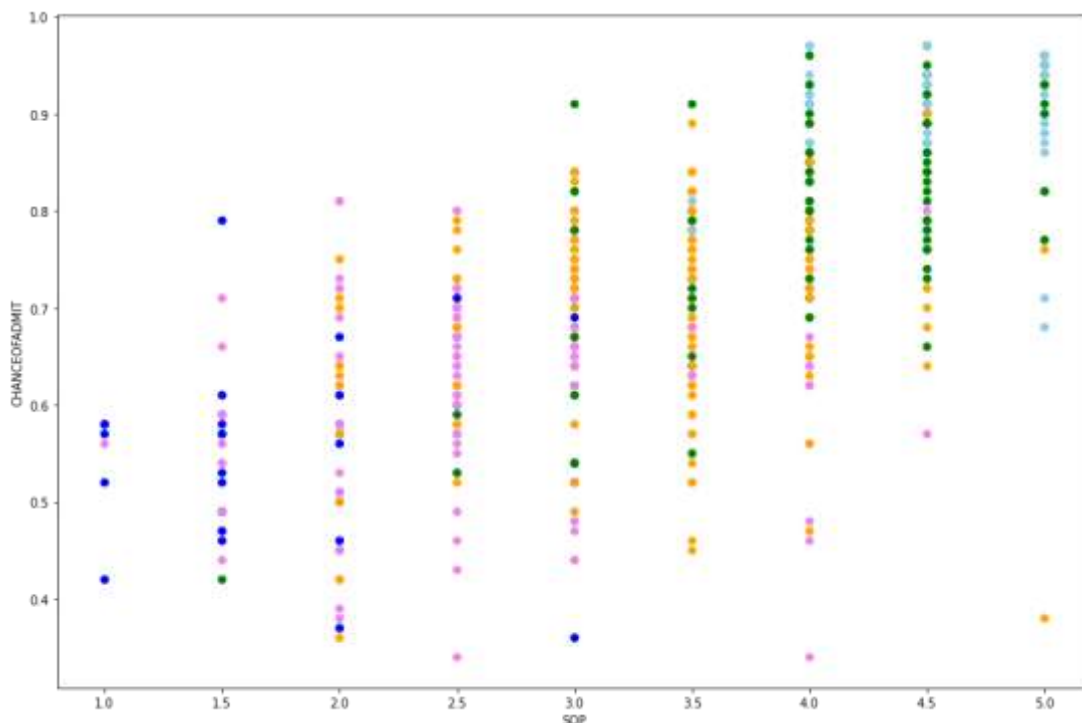


Los puntos celestes, que se ubican en lo mas alto en cuanto chances de admisión, corresponden a aquellos alumnos que desean ingresar a universidades con una calificación de 5 puntos (la mas alta). En cuanto a los puntos azules, que se ubican por debajo en su mayoría, representan a los alumnos que desean ingresar a una universidad con calificación de 1 punto.

Se puede observar mejor en un gráfico entre la chance de admisión y el puntaje CGPA, por ejemplo:



Por otro lado, quienes obtienen un mejor puntaje en su declaración de propósito, también son quienes aspiran a ingresar a las universidades mas prestigiosas. Otro indicador mas de que a mayor prestigio de la institución, mayor es el esfuerzo de los alumnos.



Una vez vistas las correlaciones, es momento de hacer la regresión lineal sobre las variables para poder predecir el resultado que tendrá CHANCEOFADMIT en base a las calificaciones obtenidas. Para esto primero hay que definir los ejes X e Y:

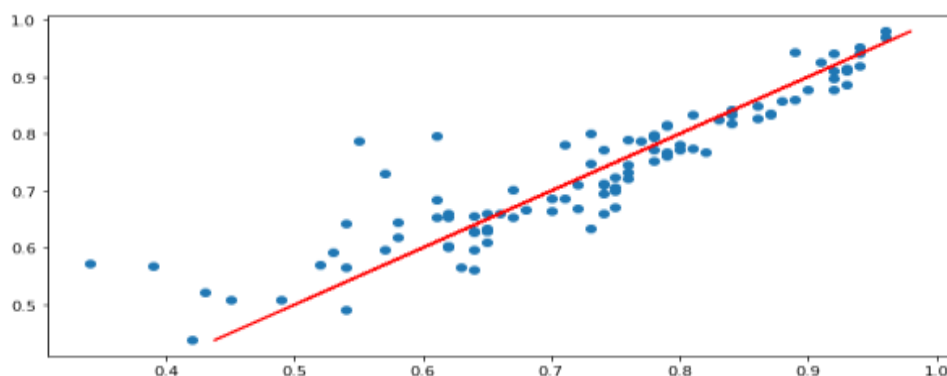
```
#Asignar las variables x e y:
y = df['CHANCEOFADMIT']
x = df.drop(['CHANCEOFADMIT', 'SERIALNUMBER'], axis=1)
```

Luego, definir un conjunto de entrenamiento y de prueba para el modelo, tomamos en este caso el 20% del total de los datos para hacer la prueba y el resto para el entrenamiento. La función `LinearRegression()` ajusta un modelo lineal para minimizar la suma residual de cuadrados entre los objetivos observados en el conjunto de datos y los objetivos predichos por la aproximación lineal, luego con `LinearRegression.fit()` entrenamos el modelo y con `LinearRegression.predict()` comprobamos los resultados:

```
#Dividir en conjunto de entrenamiento y prueba:
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=1, shuffle=True)
regression_lineal = LinearRegression()
regression_lineal.fit(x_train, y_train)
y_pred = regression_lineal.predict(x_test)
```

Mediante la función `matplotlib.pyplot.scatter()` se obtiene un gráfico de dispersión entre X e Y. Luego, mediante `matplotlib.pyplot.plot()` la recta que se ajusta a la regresión lineal, quedando de este modo:

```
#Gráfico de dispersión:
plt.figure(figsize = (10,5))
plt.scatter(y_test, y_pred)
plt.plot(y_pred, y_pred, color='red')
plt.show()
```



Mediante `r2_score()` medimos que tan cerca están los datos reales de la regresión ajustada y mediante `mean_squared_error()` obtenemos la raíz del error cuadrático medio:

```
from sklearn.metrics import accuracy_score, r2_score,
mean_squared_error
#Calculamos el coeficiente de determinación:
r2 = r2_score(y_test, y_pred)
print(f"el coef de determinacion es: {r2}")
#Calculamos el error cuadrático medio:
mse = mean_squared_error(y_test, y_pred)
print("la raíz del error cuadrático medio es: ", np.sqrt(mse))
```

```
el coef de determinacion es: 0.8208741703103734
la raíz del error cuadrático medio es: 0.05881410457650766
```

Se obtuvo un valor de 0.82 para  $R^2$  lo que indica que el modelo predice la chance de admisión con un 82% de efectividad. El resultado podría ser mejor, pero es un valor aceptable, obtener un modelo casi perfecto es difícil de conseguir. En cuanto al valor de la raíz del error cuadrático medio de aproximadamente 0.059, lo que explica la diferencia entre los valores previstos y los valores observados, el valor obtenido es un buen valor.

Ahora que el modelo ya está listo, hay que probar su funcionamiento mediante el ingreso de nuevos datos:

```
prueba = ({'GRE': [337,324,316,322,314], 'TOEFL': [118,107,104,110,103],
'UNIVERSITYRATING': [4,4,3,3,2], 'SOP': [4.5,4,3,3.5,2], 'LOR':
[4.5,4.5,3.5,2.5,3], 'CGPA': [9.65,8.87,8,8.67,8.21], 'RESEARCH': [1,1,1,1,0]})
df_prueba = pd.DataFrame(prueba, index=[0,1,2,3,4])
predicted_values = regresion_lineal.predict(df_prueba)
predicted_values

array([0.95230953, 0.80104525, 0.65177204, 0.74707669, 0.63320441])
```

Los datos aportados para la prueba son los mismos que aparecen en las primeras 5 filas del dataframe, por lo que se puede comparar el resultado obtenido por el modelo y el resultado real, por ejemplo, para la primera fila el resultado es de 95% de chances de ingresar, mientras que el resultado inicial era de 92%, una diferencia de 3%. Para la segunda fila de datos se obtuvo 80% frente al 76% inicial. Para la tercera fila 65% frente a 72%. Y para las filas 4 y 5, 74% y 63% frente a los 80% y 65% iniciales, respectivamente.

El modelo no predice con exactitud, pero se acerca a los valores iniciales, dando la mejor predicción posible para los datos con los que se cuentan.

Se obtienen más datos utilizando la librería statsmodel y su función OLS, que utiliza los mínimos cuadrados ordinarios para estimar la regresión lineal, se puede visualizar, por ejemplo, el valor de cada coeficiente dentro de la regresión, y el p-value de c/u.

```
import statsmodels.api as sm
x_train = sm.add_constant(x_train, prepend=True)
modelo = sm.OLS(endog=y_train, exog=x_train,)
modelo = modelo.fit()
print(modelo.summary())
```

OLS Regression Results

```
=====
Dep. Variable:          CHANCEOFADMIT      R-squared:                0.822
Model:                  OLS                Adj. R-squared:           0.818
Method:                 Least Squares      F-statistic:             257.7
Date:                  Mon, 03 Jul 2023    Prob (F-statistic):      2.10e-142
Time:                  13:49:01           Log-Likelihood:          559.27
No. Observations:      400               AIC:                    -1103.
Df Residuals:          392               BIC:                    -1071.
Df Model:              7
Covariance Type:       nonrobust
=====
```

	coef	std err	t	P> t	[0.025	0.975]
const	-1.2887	0.118	-10.890	0.000	-1.521	-1.056
GRE	0.0018	0.001	3.135	0.002	0.001	0.003
TOEFL	0.0032	0.001	3.156	0.002	0.001	0.005
UNIVERSITYRATING	0.0061	0.004	1.387	0.166	-0.003	0.015
SOP	0.0030	0.005	0.591	0.555	-0.007	0.013
LOR	0.0144	0.005	3.105	0.002	0.005	0.024
CGPA	0.1167	0.011	10.743	0.000	0.095	0.138
RESEARCH	0.0199	0.007	2.668	0.008	0.005	0.035

```
=====
Omnibus:                80.594      Durbin-Watson:           1.932
Prob(Omnibus):          0.000      Jarque-Bera (JB):        167.116
Skew:                  -1.064      Prob(JB):                5.14e-37
Kurtosis:              5.346      Cond. No.:               1.31e+04
=====
```

Se pueden observar, ahora, datos adicionales como los p-values de cada variable, en los casos de SOP y UNIVERSITYRATING son mayores que 0,05 por lo que no podemos rechazar la hipótesis nula y decir que estas variables son explicativas del comportamiento de CHANCEOFADMIT, por lo que podríamos sacarlas de la función de regresión, aunque no lo haremos esta vez.

### Análisis de homocedasticidad:

La homocedasticidad es una característica de un modelo de regresión lineal que implica que la varianza de los errores es constante a lo largo del tiempo.

Este término, que es lo contrario de heterocedasticidad, se emplea para nombrar la propiedad de algunos modelos de regresión lineal en los que los errores de estimación son constantes a lo largo de las observaciones. Una varianza constante nos permite disponer de modelos más fiables. Además, si una varianza, aparte de ser constante es también más pequeña, nos dará como resultado una predicción del modelo más fiable.

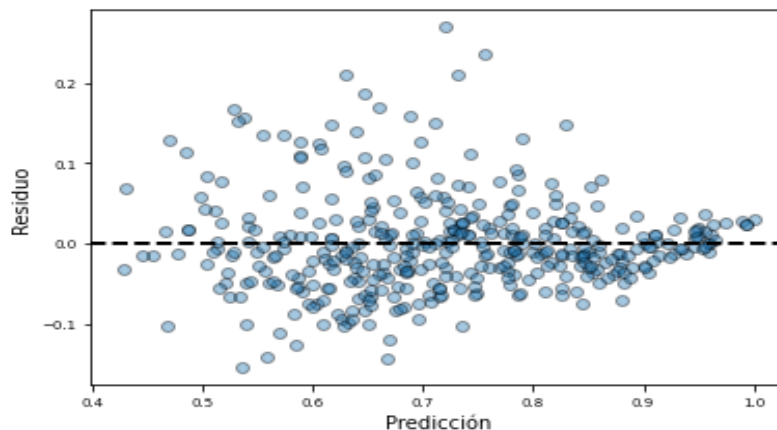
Fuente: <https://economipedia.com/definiciones/homocedasticidad.html>



```

prediccion_train = modelo.predict(exog = x_train)
residuos_train = prediccion_train - y_train
plt.scatter(prediccion_train, residuos_train,
            edgecolors=(0, 0, 0), alpha = 0.4)
plt.axhline(y = 0, linestyle = '--', color = 'black', lw=2)
plt.xlabel('Predicción')
plt.ylabel('Residuo')
plt.tick_params(labelsize = 7)

```



Al comparar los residuos con los valores otorgados por la predicción, se aprecia que no hay una distribución aleatoria en torno a cero. Esto puede indicar a que hay falta de homocedasticidad en el modelo.

Por medio del test estadístico Shapiro-Wilk vemos si podemos descartar la hipótesis nula que considera que los datos siguen una distribución normal:

```

from scipy import stats
shapiro_test = stats.shapiro(residuos_train)
shapiro_test

ShapiroResult(statistic=0.9109652042388916, pvalue=1.3475717361378838e-14)

```

El p-value obtenido es menor que 0.05 por lo que se puede rechazar la hipótesis nula. No podemos asegurar que los datos sigan una distribución normal, por lo tanto, no hay homocedasticidad en el modelo.

## Conclusiones:

El modelo de regresión lineal obtenido es el siguiente:

$$CHANCEOFADMIT = -1,288725 + 0,001832.GRE + 0,003174.TOEFL + 0,006129.UNIVERSITYRATING + 0,003005.SOP + 0,014427.LOR + 0,116704.CGPA + 0,019890.RESEARCH$$

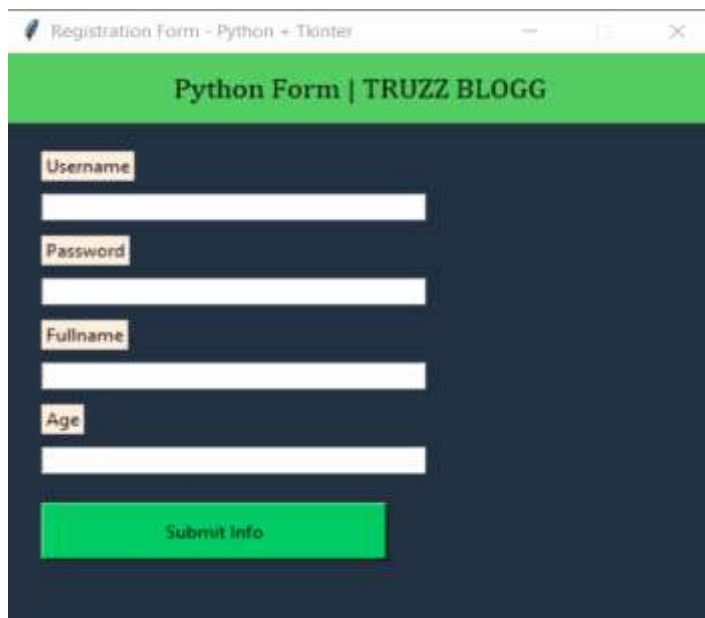
Explica el 82.2% de la varianza observada en la chance de admisión. La raíz del error cuadrático medio es  $\approx 0.0587$

No se satisfacen las condiciones de normalidad, por lo que los intervalos de confianza estimados para los coeficientes y las predicciones no son fiables.

## EJEMPLO DE PUESTA EN PRÁCTICA

Si se precisa, por ejemplo, obtener el resultado de la predicción reflejado de forma visual para el usuario final, es decir los alumnos, podría aplicarse a una página web donde estos ingresen las calificaciones obtenidas y obtengan una predicción de las chances que tienen de ingresar a la universidad solicitada. El objetivo del proyecto no es el desarrollo web, ni cuento con los conocimientos necesarios para llevar a cabo esa tarea, pero se propone un ejemplo visual, a través de una interfaz gráfica, de como funcionaria el modelo. Esto puede ser aplicado, luego, a una página web si se quiere.

El primer paso consistió en hacer una búsqueda de un ejemplo que se acerque a lo requerido para el proyecto. El enlace de donde se sacó el ejemplo es el siguiente: [https://github.com/tbcodes/registration\\_form\\_python\\_tkinter/blob/master/registration\\_form\\_python\\_tkinter.py](https://github.com/tbcodes/registration_form_python_tkinter/blob/master/registration_form_python_tkinter.py)



*Así se ve la interfaz elegida, permite ingresar datos del usuario y los guarda en un archivo csv.*

Código original de la interfaz:

```
from tkinter import *
#import tkinter as tk
#Manipulate data from registration fields
def send_data():
    username_info = username.get()
    password_info = password.get()
    fullname_info = fullname.get()
    age_info = str(age.get())
    print(username_info, "\t", password_info, "\t", fullname_info, "\t", age_info)
# Open and write data to a file
file = open("user.txt", "a")
```

```

file.write(username_info)
file.write("\t")
file.write(password_info)
file.write("\t")
file.write(fullname_info)
file.write("\t")
file.write(age_info)
file.write("\t\n")
file.close()

print("    New user registered. Username: {} | FullName:
{} ").format(username_info, fullname_info))
# Delete data from previous event
username_entry.delete(0, END)
password_entry.delete(0, END)
fullname_entry.delete(0, END)
age_entry.delete(0, END)
# Create new instance - Class Tk()
mywindow = Tk()
mywindow.geometry("450x450")
mywindow.title("Registration Form - Python + Tkinter")
mywindow.resizable(False, False)
mywindow.config(background = "#213141")
main_title = Label(text = "Python Form | TRUZZ BLOGG", font = ("Cambria", 14),
bg = "#56CD63", fg = "black", width = "500", height = "2")
main_title.pack()
# Define Label Fields
username_label = Label(text = "Username", bg = "#FFEEDD")
username_label.place(x = 22, y = 70)
password_label = Label(text = "Password", bg = "#FFEEDD")
password_label.place(x = 22, y = 130)
fullname_label = Label(text = "Fullname", bg = "#FFEEDD")
fullname_label.place(x = 22, y = 190)
age_label = Label(text = "Age", bg = "#FFEEDD")
age_label.place(x = 22, y = 250)
# Get and store data from users
username = StringVar()
password = StringVar()
fullname = StringVar()
age = StringVar()
username_entry = Entry(textvariable = username, width = "40")
password_entry = Entry(textvariable = password, width = "40", show = "*")
fullname_entry = Entry(textvariable = fullname, width = "40")
age_entry = Entry(textvariable = age, width = "40")
username_entry.place(x = 22, y = 100)
password_entry.place(x = 22, y = 160)
fullname_entry.place(x = 22, y = 220)
age_entry.place(x = 22, y = 280)
# Submit Button

```

```

submit_btn = Button(mywindow, text = "Submit Info", width = "30", height = "2",
command = send_data, bg = "#00CD63")
submit_btn.place(x = 22, y = 320)
mywindow.mainloop()

```

Los cambios realizados permiten, ahora, ingresar los datos de las calificaciones obtenidas y obtener la probabilidad de ingresar a la institución, el código final queda así:

```

from tkinter import *
from tkinter import ttk
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, r2_score, mean_squared_error
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
# Manipulate data from registration fields
def usar_data():
    gre_info = float(gre.get())
    toefl_info = float(toefl.get())
    cgpa_info = float(cgpa.get())
    #cargamos el dataset jamboree
    ruta =
r'C:/Users/matia/OneDrive/Escritorio/Curso_The_Corner/jamboree/jamboree_datase
t.csv'
    df = pd.read_csv(ruta)
    #cambiamos el nombre de las columnas
    df = df.rename(columns={'Serial No.': 'SERIALNUMBER', 'GRE Score': 'GRE',
'TOEFL Score': 'TOEFL', 'University Rating': 'UNIVERSITYRATING',
'Research': 'RESEARCH', 'Chance of Admit ': 'CHANCEOFADMIT'})
    #asignar las variables x e y:
    y = df['CHANCEOFADMIT']
    x = df.drop(['CHANCEOFADMIT', 'SERIALNUMBER'], axis=1)
    # dividir en conjunto de entrenamiento y prueba:
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=1, shuffle=True)
    regresion_lineal = LinearRegression()
    regresion_lineal.fit(x_train, y_train)
    y_pred = regresion_lineal.predict(x_test)
    gre_valor = gre_info

```

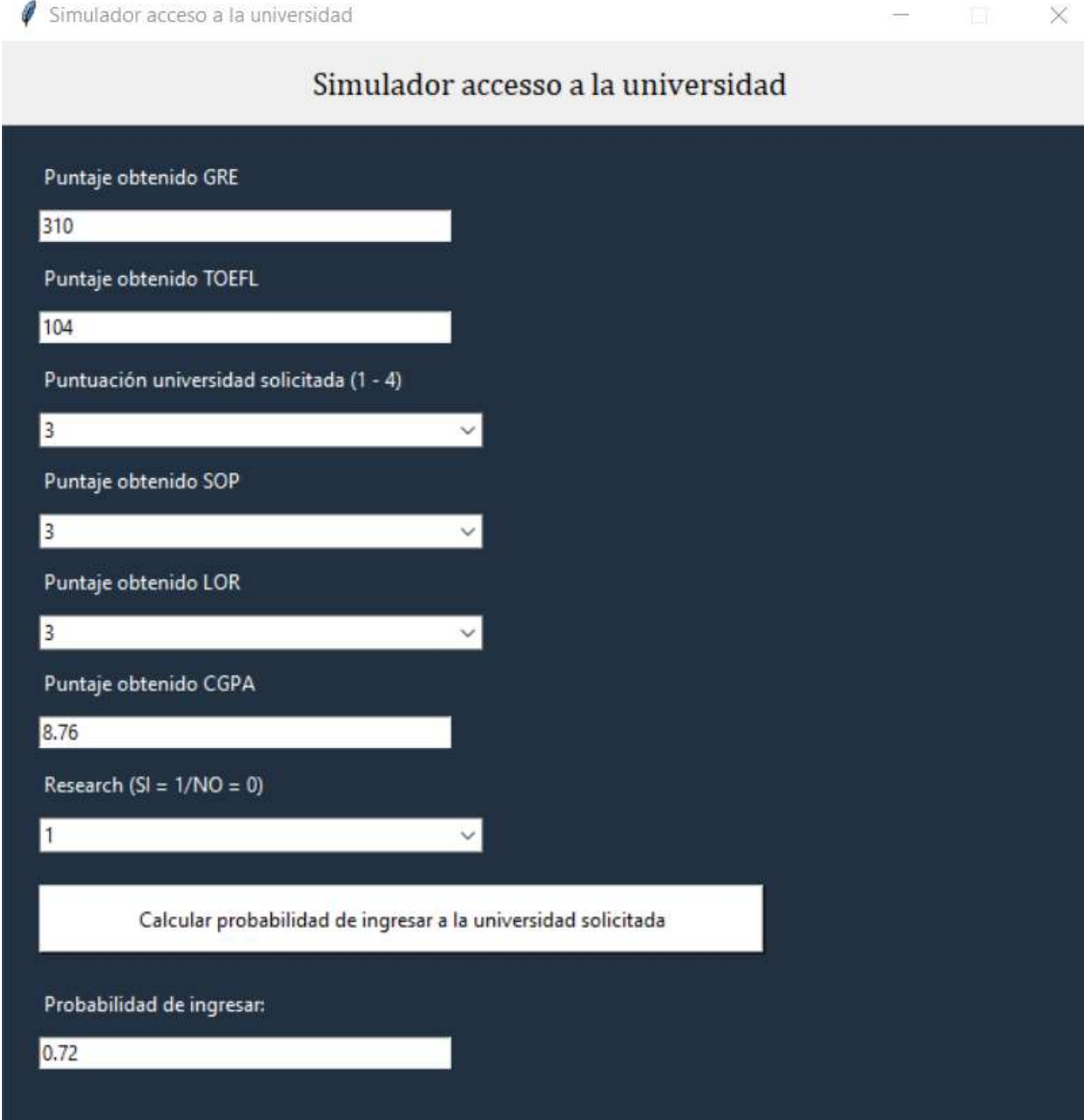
```

toefl_valor = toefl_info
cgpa_valor = cgpa_info
prueba = ({'GRE': [gre_valor], 'TOEFL': [toefl_valor], 'UNIVERSITYRATING':
[universidadrating_entry.get()], 'SOP': [sop_entry.get()], 'LOR ':
[lor_entry.get()], 'CGPA': [cgpa_valor], 'RESEARCH': [research_entry.get()]})
df_prueba = pd.DataFrame(prueba, index=[0])
predicted_values = regresion_lineal.predict(df_prueba)
#print("\nposibilidad de ingresar a la universidad solicitada:
",predicted_values)
if (gre_valor.is_integer() == True) and (toefl_valor.is_integer() == True)
and (0<=cgpa_valor<=10):
    resultado.set(round(predicted_values[0], 2))
else:
    resultado.set("Error, revise los datos ingresados")
#creamos la etiqueta que arroja el valor predict
# Create new instance - Class Tk()
mywindow = Tk()
mywindow.geometry("800x800")
mywindow.title("Simulador acceso a la universidad")
mywindow.resizable(False,False)
mywindow.config(background = "#213141")
main_title = Label(text = "Simulador acceso a la universidad", font =
("Cambria", 14), fg = "black", width = "500", height = "2")
main_title.pack()
# Define Label Fields
gre_label = Label(text = "Puntaje obtenido GRE", bg = "#213141",
foreground='white')
gre_label.place(x = 22, y = 70)
toefl_label = Label(text = "Puntaje obtenido TOEFL", bg = "#213141",
foreground='white')
toefl_label.place(x = 22, y = 130)
universidadrating_label = Label(text = "Puntuación universidad solicitada (1 -
4)", bg = "#213141", foreground='white')
universidadrating_label.place(x = 22, y = 190)
sop_label = Label(text = "Puntaje obtenido SOP", bg = "#213141",
foreground='white')
sop_label.place(x = 22, y = 250)
lor_label = Label(text = "Puntaje obtenido LOR", bg = "#213141",
foreground='white')
lor_label.place(x = 22, y = 310)
cgpa_label = Label(text = "Puntaje obtenido CGPA", bg = "#213141",
foreground='white')
cgpa_label.place(x = 22, y = 370)
research_label = Label(text = "Research (SI = 1/NO = 0)", bg = "#213141",
foreground='white')
research_label.place(x = 22, y = 430)
resultado_label = Label(text = "Probabilidad de ingresar: ", bg = "#213141",
foreground='white')
resultado_label.place(x = 22, y = 560)

```

```
# Get and store data from users
gre = StringVar()
toefl = StringVar()
cgpa = StringVar()
resultado = StringVar()
gre_entry = Entry(textvariable = gre, width = "40")
toefl_entry = Entry(textvariable = toefl, width = "40")
universidadrating_entry = ttk.Combobox(state="readonly", values=[1, 2, 3, 4],
width="40")
sop_entry = ttk.Combobox(state="readonly", values=[1, 1.5, 2, 2.5, 3, 3.5, 4,
4.5, 5], width="40")
lor_entry = ttk.Combobox(state="readonly", values=[1, 1.5, 2, 2.5, 3, 3.5, 4,
4.5, 5], width="40")
cgpa_entry = Entry(textvariable = cgpa, width = "40")
research_entry = ttk.Combobox(state="readonly", values=[0, 1], width="40")
resultado_entry = Entry(textvariable = resultado, width = "40")
gre_entry.place(x = 22, y = 100)
toefl_entry.place(x = 22, y = 160)
universidadrating_entry.place(x = 22, y = 220)
sop_entry.place(x = 22, y = 280)
lor_entry.place(x = 22, y = 340)
cgpa_entry.place(x = 22, y = 400)
research_entry.place(x = 22, y = 460)
resultado_entry.place(x = 22, y = 590)
# Submit Button
submit_btn = Button(mywindow, text = "Calcular probabilidad de ingresar a la
universidad solicitada", width = "60", height = "2", command = usar_data, bg =
"white")
submit_btn.place(x = 22, y = 500)
mywindow.mainloop()
```

Y visualmente obtenemos lo siguiente:



Simulador acceso a la universidad

Puntaje obtenido GRE  
310

Puntaje obtenido TOEFL  
104

Puntuación universidad solicitada (1 - 4)  
3

Puntaje obtenido SOP  
3

Puntaje obtenido LOR  
3

Puntaje obtenido CGPA  
8.76

Research (SI = 1/NO = 0)  
1

Calcular probabilidad de ingresar a la universidad solicitada

Probabilidad de ingresar:  
0.72

Este es un ejemplo sencillo de cómo se puede aplicar el análisis de la regresión lineal a una pagina web para que los alumnos puedan ingresar los resultados y obtener su chance de ser admitidos.

## ANÁLISIS EXTRA:

Una idea inicial al comenzar con el proyecto fue realizar una función que defina si un alumno ingresa o no a la universidad, en base a la probabilidad inicial que le concedían las calificaciones obtenidas, entrenando un modelo para predecir futuros casos.

El primer paso es, crear una columna que diga si el alumno ha sido admitido o no, como este resultado será dado en base a la probabilidad descrita en la columna CHANCEOFADMIT, dependerá de los valores de las otras columnas también.

```
#Creamos la columna ADMIT para definir si la persona fue admitida o no en la
universidad.
#Esto sirve para predecir el resultado con nuevos ingresos de datos.
#Luego eliminamos CHANCEOFADMIT ya que no continuaremos utilizándola.
admitidos = []
for i in df['CHANCEOFADMIT'].values:
    opciones = [1, 0]
    probabilidades = [i, 1-i]
    eleccion = np.random.choice(opciones, p=probabilidades)
    admitidos.append(eleccion)
df['ADMIT'] = admitidos
df.head()
```

	SERIALNUMBER	GRE	TOEFL	UNIVERSITYRATING	SOP	LOR	CGPA	RESEARCH	CHANCEOFADMIT	ADMIT
0	1	337	118	4	4.5	4.5	9.65	1	0.92	1
1	2	324	107	4	4.0	4.5	8.87	1	0.76	0
2	3	316	104	3	3.0	3.5	8.00	1	0.72	1
3	4	322	110	3	3.5	2.5	8.67	1	0.80	1
4	5	314	103	2	2.0	3.0	8.21	0	0.65	1

Luego, se elimina la columna CHANCEOFADMIT, después de todo, el objetivo es crear un modelo que haga una predicción de si el alumno es admitido o no, y está este determinada por los valores de las calificaciones obtenidas.

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

Mediante df.value\_counts() se ve que cantidad de alumnos fueron admitidos y que cantidad no (361 si lo fueron y 139 no):

```
df['ADMIT'].value_counts()
```

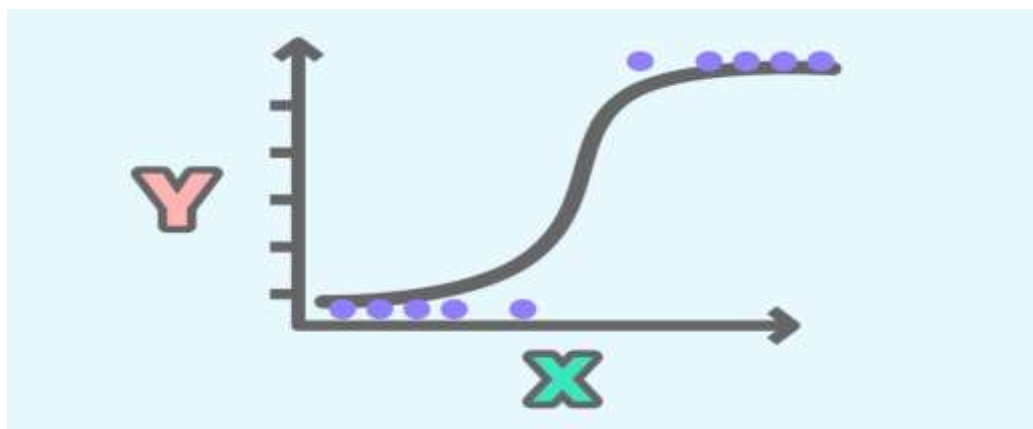
```
1    361
0    139
Name: ADMIT, dtype: int64
```



Para predecir si un alumno será admitido o no, en base a los resultados obtenidos en las diferentes evaluaciones, se utiliza la regresión lineal con la función `LogisticRegression()` de `sklearn`, para esto primero se escalan las distintas variables con la función `MinMaxScaler()` también de `sklearn`, luego se dividen los datos en modelos de prueba y entrenamiento con `train_test_split`, como se hizo anteriormente.

A continuación, una breve descripción de regresión logística:

*En estadística, la regresión logística es un tipo de modelo de regresión que sirve para predecir el resultado de una variable categórica. Es decir, la regresión logística se usa para modelar la probabilidad de que una variable categórica tome un determinado valor en función de las variables independientes.*



Fuente: <https://www.probabilidadyestadistica.net/regresion-logistica/>

```
#Vamos a entrenar un modelo para predecir si una persona será admitida o no en
base a los datos ingresados
#En las demás columnas. Para esto debemos tomar valores de pruebas.
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
y = df['ADMIT']
df = df.drop(['ADMIT'], axis=1)
scaler = MinMaxScaler()
X = scaler.fit_transform(df)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
#vemos la precision:
print("Accuracy {} %".format(100*accuracy_score(y_pred, y_test)))
```

Accuracy 79.0 %

La regresión predice los resultados con un 79% de exactitud, se considera un resultado un poco aceptable, aunque se debe ver en más detalle el comportamiento del modelo.

```
X_train.shape
```

```
(400, 8)
```

```
X_test.shape
```

```
(100, 8)
```

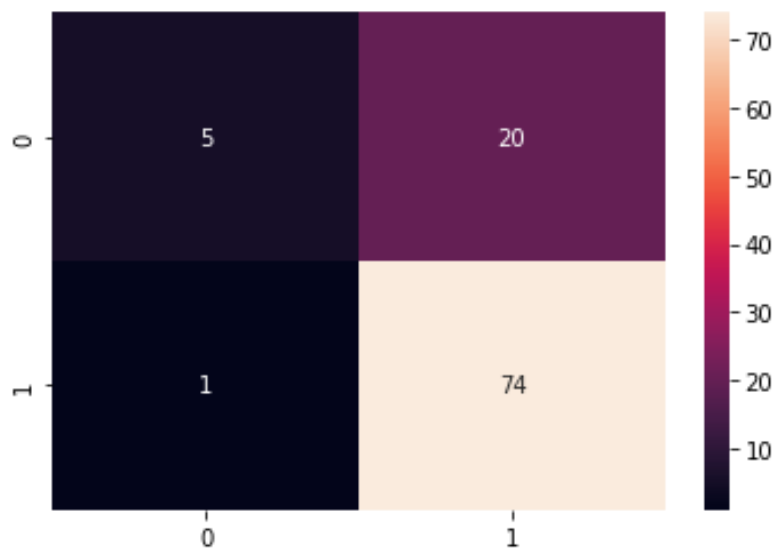
Se utiliza un conjunto de entrenamiento de 400 filas y uno de prueba de 100, luego a través de una matriz de confusión se ven los resultados:

```
#Resultados en el Conjunto de Testing
```

```
cm = confusion_matrix( y_test, y_pred)
```

```
sns.heatmap(cm, annot=True)
```

```
<AxesSubplot:>
```



```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.83	0.20	0.32	25
1	0.79	0.99	0.88	75
accuracy			0.79	100
macro avg	0.81	0.59	0.60	100
weighted avg	0.80	0.79	0.74	100

Explicación de la matriz de confusión y los resultados obtenidos:

La matriz de confusión es una herramienta muy útil para valorar cómo de bueno es un modelo de clasificación basado en aprendizaje automático. En particular, sirve para mostrar de forma explícita cuándo una clase es confundida con otra, lo cual nos permite trabajar de forma separada con distintos tipos de error.

Fuente: <https://telefonicatech.com/blog/como-interpretar-la-matriz-de-confusion-ejemplo-practico>

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

En el gráfico se observa los resultados obtenidos por la predicción y los resultados reales, siendo uno verdadero (admitido en este caso) y cero falso (no admitido). Luego:

TN = TRUE NEGATIVE (verdadero negativo)

FP = FALSE POSITIVE (falso positivo)

FN = FALSE NEGATIVE (falso negativo)

TP = TRUE POSITIVE (verdadero positivo)

En cuanto a los resultados observados en el ejemplo, dados por la función `classification_report()`, se ven más detallados a continuación:

$$\text{precisión} = \frac{TP}{TP + FP} = \frac{74}{74 + 20} \approx 0,79$$

$$\text{recall} = \frac{TP}{TP + FN} = \frac{74}{74 + 1} \approx 0,99$$

$$F1 = 2 \cdot \frac{\text{precisión} \cdot \text{recall}}{\text{precisión} + \text{recall}} \approx 0,88$$

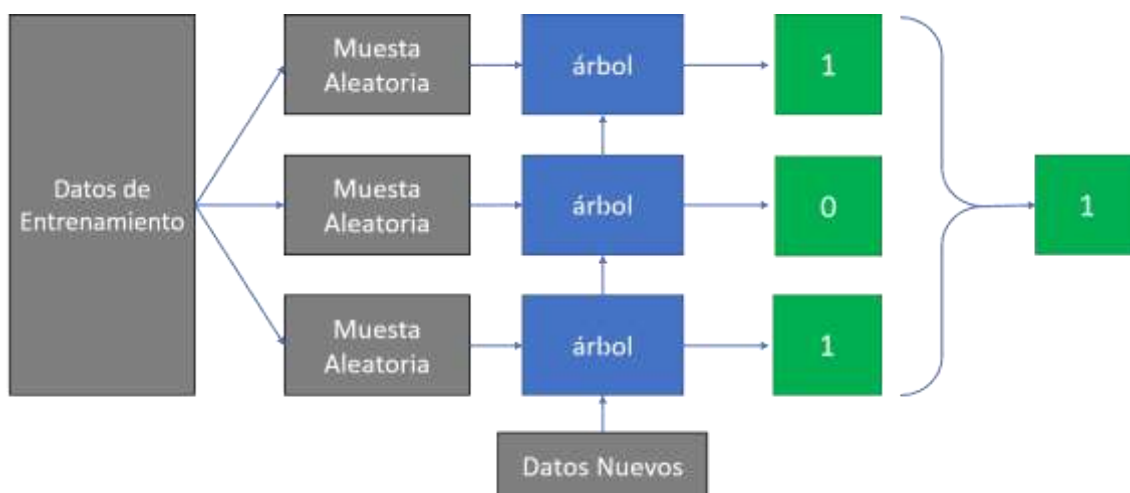
$$\text{accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \frac{74 + 5}{74 + 5 + 20 + 1} = 0,79$$

$$\text{especificidad} = \frac{TN}{TN + FP} = \frac{5}{5 + 20} = 0,2$$

Si bien los valores obtenidos en base a los casos positivos son buenos, al calcular la especificidad, es decir, la tasa de acierto que el algoritmo tiene con respecto a los casos negativos se observa que hay muchos casos que se dan por positivos cuando en realidad son negativos. Al final, el algoritmo da muchos mas casos positivos de los que realmente hay. Es por esto último que la decisión es no continuar con este análisis.

Otra opción: utilizar random forest en vez de regresión logística:

Un Random Forest es un conjunto de árboles de decisión combinados con bagging. Al usar bagging, lo que en realidad está pasando, es que distintos árboles ven distintas porciones de los datos. Ningún árbol ve todos los datos de entrenamiento. Esto hace que cada árbol se entrene con distintas muestras de datos para un mismo problema. De esta forma, al combinar sus resultados, unos errores se compensan con otros y tenemos una predicción que generaliza mejor.



Fuente: <https://www.iartificial.net/random-forest-bosque-aleatorio/>

El resultado final es, entonces, el que se obtuvo en la mayoría de arboles de decisión. En el ejemplo de la imagen se obtuvieron dos resultados igual a uno, y un resultado igual a cero, entonces uno es la mayoría por lo tanto es el resultado escogido.

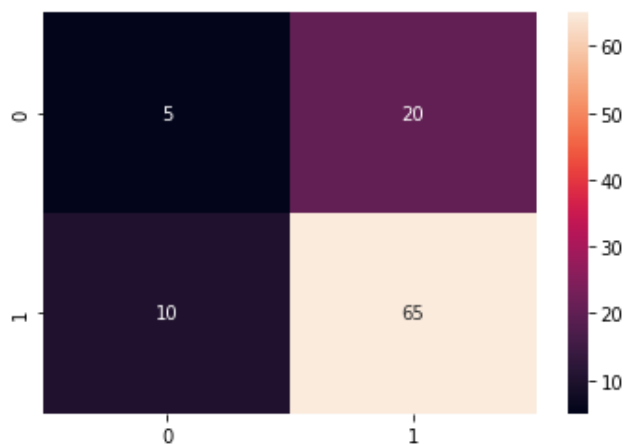
```

model = RandomForestClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# Resultados en el Conjunto de Testing
cm = confusion_matrix( y_test, y_pred)
sns.heatmap(cm, annot=True)

```

<AxesSubplot:>



```
print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.33	0.20	0.25	25
1	0.76	0.87	0.81	75
accuracy			0.70	100
macro avg	0.55	0.53	0.53	100
weighted avg	0.66	0.70	0.67	100

El resultado obtenido es aún menos efectivo que utilizando la regresión logística, se puede observar en la matriz que hay más falsos negativos.

En conclusión, al no obtener un resultado satisfactorio para el análisis, no se continúa con el modelo. Permitted, de todas maneras, explorar más sobre las librerías de Python aplicadas al machine learning y el análisis de datos, por lo que se incluye en el proyecto, al final, el objetivo de el mismo desde un principio fue explorar y aprender sobre esta disciplina.

Se da por finalizado el proyecto, sirviendo de base para continuar con el aprendizaje, seguro que hay mucho por mejorar y se espera volver aquí las veces que sean necesarias para enriquecer los conocimientos.