



**Universidad  
Nacional de  
General  
Sarmiento**

**TRABAJO PRÁCTICO**  
**PROYECTO PROFESIONAL I**  
**MACHINE LEARNING**

**Materia:** Proyecto profesional I

**Comisión:** 01 - 2/2022

**Docentes:** Juan Carlos Monteros, Francisco Orozco De La Hoz, Leandro Dikenstein

**Integrantes:** Lautaro Luque Materazzi, Alexis Valentín Richter

# ÍNDICE

<b>Investigación sobre ML</b>	<b>4</b>
<b>Selección de usos</b>	<b>4</b>
<b>Plataforma</b>	<b>5</b>
<b>Pasos para el desarrollo del TP</b>	<b>5</b>

# Investigación sobre ML

Indagando y leyendo la documentación brindada por la materia, más nuestra propia investigación en internet, fuimos teniendo más noción sobre Machine Learning. En primer lugar, nos dimos cuenta de la gran cantidad de ejemplos y cursos que hay para analizar y profundizar.

Si bien todavía no tenemos un conocimiento completo de todo lo que implica Machine Learning, decidimos que dado los modelos propuestos en la materia, los más interesantes y a su vez, los que mayor información hay, son el de Predicción, Clasificación y Crear Cluster a partir de datos sin etiquetar. Los primeros dos entran en la categoría de aprendizaje supervisado mientras que el último es no supervisado.

## Selección de usos

1. Predicción
2. Clasificación
3. Crear Clusters a partir de datos sin etiquetar

1) **Predicción:** Para este modelo, nos pareció interesante la posibilidad de poder predecir diabetes en una persona a través de la edad, sexo, glucosa, IMC, etc. Para eso aplicaremos el algoritmo RandomForest, puesto que se adapta muy bien para este tipo de modelos y hay mucha documentación al respecto. El dataset será a través de la librería sklearn..

2) **Clasificación:** en cuanto a la clasificación el objetivo es poder clasificar imágenes números escritos a mano por diferentes personas, usaremos el dataset MNIST que proporciona la librería Keras. En cuanto al algoritmo que usaremos será el de Convolutional Neural Network ya que investigando este algoritmo es uno de los mejores para interpretación de imágenes. El cual es

de Aprendizaje supervisado (clasificación) y también es proporcionado por Keras.

- 3) **Crear clusters a partir de datos sin etiquetar:** Utilizaremos para este modelo el algoritmo K-means Clustering con el objetivo de poder etiquetar los diferentes tipos de flores iris (setosa, virginica, versicolor) a través de las características como el largo y ancho del sépalo y pétalo. Para esto se va a usar el dataset de iris que proporciona la librería sklearn en donde los datos ya están normalizados y listos para implementar.

## Plataforma

En cuanto a la plataforma, contemplamos la posibilidad de utilizar Google Colab, ya que nos llamó la atención que se pueda trabajar desde la nube y con el beneficio de tener Jupyter Notebook y varias librerías preinstaladas. Pero también observamos algunos limitantes como la posibilidad del manejo de archivos, que caducan a las 24 hs. Debido a esto, finalmente utilizaremos Anaconda y Jupyter Notebook de manera local utilizando la librería de python Scikit-learn, ya que es una de las más utilizadas, junto con otras como Numpy, Pandas y demás.

## Pasos para el desarrollo del TP

1. Recolectar Datos
2. Preparar los datos
3. Elegir el modelo
4. Entrenar nuestra máquina
5. Evaluación
6. Parameter Tuning (configuración de parámetros)
7. Predicción o Inferencia
8. Interpretación del Modelo

## 9. Implementar el Modelo como Servicio

1)

Link al repo:

<https://github.com/LautaroLM/Machine-Learning-TP/blob/main/K-means/Kmeans%20-%20Iris%20markdown.md>

## Kmeans sobre dataset de Iris

```
In [4]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn import datasets
```

SETEO DE VARIABLES

```
In [5]: # Loading the Dataset
```

```
df = pd.read_csv('Iris.csv')
df
```

CARGA DE DATASET

```
Out[5]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species	
	0	1	5.1	3.5	1.4	0.2	Iris-setosa
	1	2	4.9	3.0	1.4	0.2	Iris-setosa
	2	3	4.7	3.2	1.3	0.2	Iris-setosa
	3	4	4.6	3.1	1.5	0.2	Iris-setosa
	4	5	5.0	3.6	1.4	0.2	Iris-setosa
	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica	
146	147	6.3	2.5	5.0	1.9	Iris-virginica	
147	148	6.5	3.0	5.2	2.0	Iris-virginica	
148	149	6.2	3.4	5.4	2.3	Iris-virginica	
149	150	5.9	3.0	5.1	1.8	Iris-virginica	

150 rows x 6 columns

SE BORRA COLUMNA ID

```
In [6]: df.drop('Id', axis=1, inplace=True) # Se borra columna ID porque no nos proporciona data útil.
```

```
In [7]: df.head() #Imprimimos las primeros 5 filas
```

```
Out[7]:
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

```
In [8]: df.tail() #Imprimimos las últimas 5 filas.
```

```
Out[8]:
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

INFO DEL DATASET

```
In [9]: df.info() #Imprimimos info importante del data set.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
#   Column          Non-Null Count  Dtype
---  ---
0   SepalLengthCm    150 non-null   float64
1   SepalWidthCm     150 non-null   float64
2   PetalLengthCm    150 non-null   float64
3   PetalWidthCm     150 non-null   float64
4   Species          150 non-null   object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

```
In [10]: df.isnull().sum()
```

```
Out[10]: SepalLengthCm    0
SepalWidthCm          0
PetalLengthCm         0
PetalWidthCm          0
Species               0
dtype: int64
```

---

Importante no tener datos nulos. Como vemos, ninguna columna tiene datos nulos.

```
In [17]: df.corr() #Para ver la relación/correlación que hay entre las diferentes columnas.
```

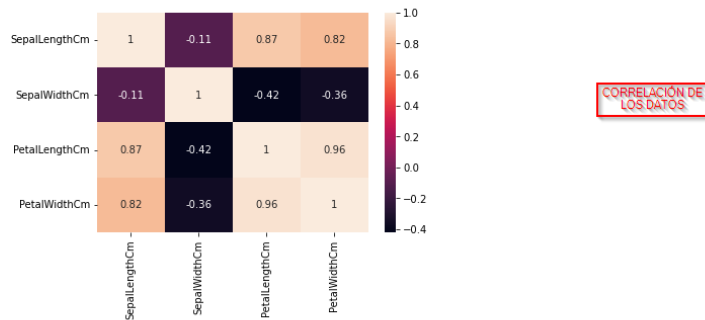
```
Out[17]:
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

En esta tabla podemos ver que la dupla que tiene la mayor relación es PetalLengthCm (largo del pétalo) con PetalWidthCm (ancho del pétalo). Tienen una correlación de 0.962757.

```
In [18]: sns.heatmap(df.corr(),annot=True ) #Otra forma de ver la relación entre las columnas pero de una manera más gráfica.
```

```
Out[18]: <AxesSubplot:~>
```



CORRELACIÓN DE LOS DATOS

Usaremos entonces el largo y ancho del pétalo.

```
In [19]: df1 = df[df['Species']=='Iris-setosa']
df2 = df[df['Species']=='Iris-versicolor']
df3 = df[df['Species']=='Iris-virginica']

plt.scatter(df1['PetalLengthCm'],df1['PetalWidthCm'], color='r' , label='Iris-setosa') # se le asigna el color rojo a setosa
plt.scatter(df2['PetalLengthCm'],df2['PetalWidthCm'], color='b' , label='Iris-versicolor') # se le asigna el color azul a versicolor
plt.scatter(df3['PetalLengthCm'],df3['PetalWidthCm'], color='g' , label='Iris-virginica') # se le asigna el color verde a virginica

plt.legend()
plt.show() #mostramos el gráfico con una leyenda.
```

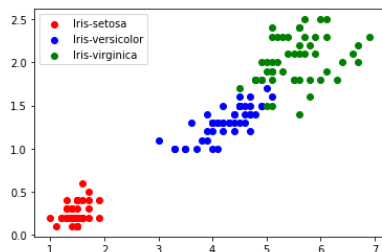


GRÁFICO DEL DATASET IRIS

Cómo se puede observar, la setosa se diferencia bastante respecto a versicolor y virginica. Estas ultimas dos, tienden a tener el pétalo algo similar.

```
In [21]: # Queremos ver cuál es el valor de K que mejor encaja con el algoritmo KMeans.
# De antemano sabemos que al ser 3 especies, el K más eficiente tiene que ser igual a 3.
```

```
df_1mp = df.iloc[:,0:4] #Nos quedamos con las primeras 4 columnas, es decir eliminamos la columna 'species'. Solo nos interesa el sépalos y pétalo.

k_meansclus = range(1,10) #Iteramos para que K tome valores de 1 a 10.
sse = []

for k in k_meansclus :
    km = KMeans(n_clusters =k)
    km.fit(df_1mp)
    sse.append(km.inertia_)

# El algoritmo tiene como objetivo elegir centroides que minimicen la inercia,
# lo que puede reconocerse como una medida de cuán coherentes internamente son los clusters.
# De esta manera, vamos a ir 'appendando' a un array los puntajes que se obtendrán para los distintos K.
# El valor que más se acerque a 100, será el más óptimo.
```

ALGORITMO PARA ENCONTRAR EL K

```
C:\Users\USUARIO\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can disable OMP_NUM_THREADS=1.
warnings.warn(
```

```
In [22]: sse #Imprimimos el array con los valores que nos devolvió el algoritmo inercia
```

```
Out[22]:
```

```
[688.8244,
152.36870647733906,
78.94084142614601,
57.34540931571816,
46.535582051282056,
38.930963049671746,
34.78153217468806,
30.011141025641027,
28.130942062323648]
```

Como podemos ver, el K más eficiente es cuando toma el valor 3, que coincide con nuestra cantidad de especies. Entonces cuando K=3, vemos que el puntaje es 78.94... , que dentro de todo es óptimo. K=2 ó K=4 ya se alejan bastante de 100.



```
In [23]: # El gráfico del codo nos dará visualmente lo que obuvimos en el array.  
plt.title('The Elbow Method')  
plt.plot(k_meansclus,sse)  
plt.show()
```

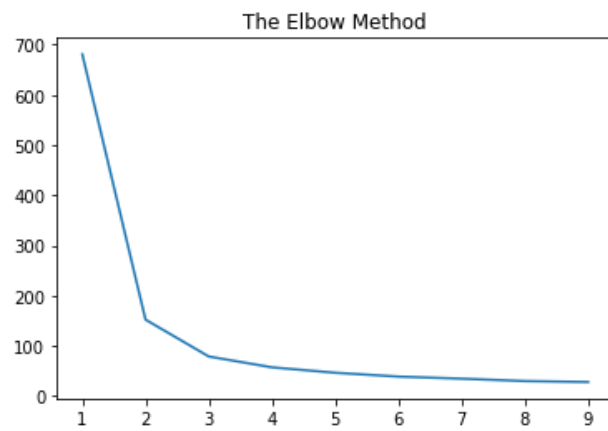
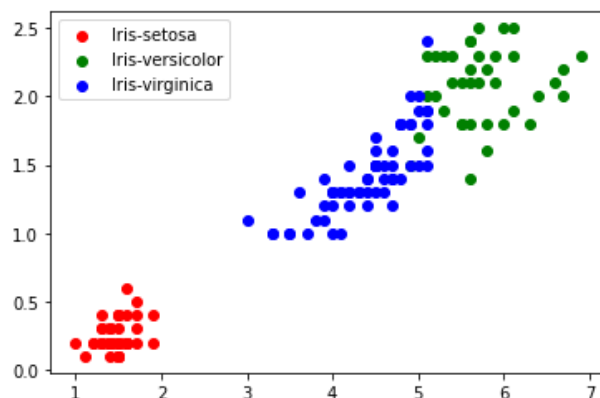


GRÁFICO DE K  
MÁS  
EFICIENTE

## ENTRENAMIENTO

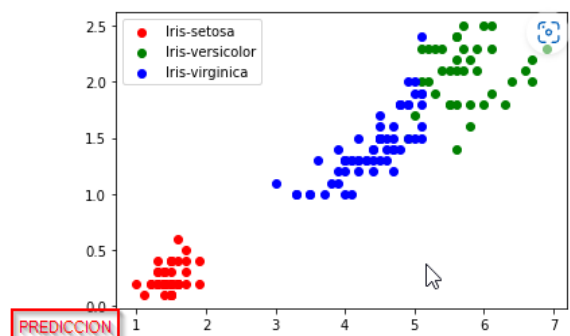
```
Out[26]: array([[5.9016129 , 2.7483871 , 4.39354839, 1.43387097],
 [5.006      , 3.418      , 1.464      , 0.244      ],
 [6.85       , 3.07368421, 5.74210526, 2.07105263]])
```

```
plt.legend()  
plt.show()
```



## PREDICCIÓN

A scatter plot showing the relationship between Sepal.Length (x-axis) and Petal.Length (y-axis) for three species of Iris flowers: Iris-setosa (red), Iris-versicolor (blue), and Iris-virginica (green). The x-axis ranges from 1 to 7, and the y-axis ranges from 0.0 to 2.5. Iris-setosa is clustered at low values for both variables. Iris-versicolor and Iris-virginica show a positive correlation, with Iris-virginica generally having higher values for both variables. A legend in the top left corner identifies the species by color.



2)

Link al repo:

<https://github.com/LautaroLM/Machine-Learning-TP/blob/main/ConvolutionalNeuralNetwork/CNN%20markdown.md>

Clasificación de números en imágenes


Importamos las librerías

```
1 import tensorflow as tf #Red neuronal convolucional
2 import pandas as pd #Visualizar data
3 import numpy as np #La librería MNIST con los datos esta en formato numpy
4 import matplotlib.pyplot as plt #Graficas
5 from tensorflow.keras.models import Sequential #Tipo de modelo secuencial, para agrupar capas
6 from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D
7 #Dense para crear capas full conectadas (RNFC)
8 #Dropout para evitar el overfitting, desactivando un porcentaje de neuronas determinado
9 #Flatten para convertir las salidas en un vector 1D
10 #Conv2D para realizar operaciones de convolucion
11 #MaxPooling para realizar operaciones de Maxpooling
```

Analizamos el set de datos

```
1 mnist_data = tf.keras.datasets.mnist
2
3 #data de entrenamiento y data de testeo
4 (train_images, train_labels), (test_images, test_labels) = mnist_data.load_data()
5
6 #este set de datos ya viene con data de entrenamiento y de test separados
7 print("Dimensiones del set de entrenamiento:", train_images.shape)
8 print("Imágenes de entrenamiento:", train_images.shape[0])
9 print("Imágenes de testeo:", test_images.shape[0])
10
11 #Dimensiones del set de entrenamiento: (60000, 28, 28)
12 #Imágenes de entrenamiento: 60000
13 #Imágenes de testeo: 10000
```

## Seteo de variables



```
1 #Variables
2
3 #cantidad de digitos a clasificar
4 num_classes = 10
5 #tamano de cada subconjunto (para no agarra las 60000 y llenar la memoria)
6 batch_size = 128
7 #cuanta veces va a recorrer todo el conjunto de entrenamiento
8 epochs = 5
9 #forma de las imagenes
10 input_shape = (28, 28, 1)
```

## Normalización de datos

```
1 #Normalizamos los valores entre 0 y 1
2 train_images = train_images.astype('float32')
3 test_images = test_images.astype('float32')
4 train_images /= 255
5 test_images /= 255
6 print("ValMinTR", np.amin(train_images))
7 print("ValMinTE", np.amin(test_images))
8 print("ValMaxTR", np.amax(train_images))
9 print("ValMaxTE", np.amax(test_images))
10 print("")
11
12 #Establecemos el numero de canales en 1 ya que la imagen esta en escala de grises
13 train_images = train_images.reshape(60000, 28, 28, 1)
14 test_images = test_images.reshape(10000, 28, 28, 1)
15 print("Dim", train_images.shape)
16 print("Dim", test_images.shape)
17 print("")
18
19 #Convertimos los vectores de clase en matrices binarias
20 print(test_labels)
21 train_labels = tf.keras.utils.to_categorical(train_labels, num_classes)
22 test_labels = tf.keras.utils.to_categorical(test_labels, num_classes)
23 print(test_labels)
24
25 # ValMinTR 0.0
26 # ValMinTE 0.0
27 # ValMaxTR 1.0
28 # ValMaxTE 1.0
29
30 # Dim (60000, 28, 28, 1)
31 # Dim (10000, 28, 28, 1)
32
33 # [7 2 1 ... 4 5 6]
34 # [[0. 0. 0. ... 1. 0. 0.]
35 #  [0. 0. 1. ... 0. 0. 0.]
36 #  [0. 1. 0. ... 0. 0. 0.]
37 #  ...
38 #  [0. 0. 0. ... 0. 0. 0.]
39 #  [0. 0. 0. ... 0. 0. 0.]
40 #  [0. 0. 0. ... 0. 0. 0.]
```

## Creación de la CNN (Red Neuronal Convolutacional)

```
1 #Creacion del modelo
2 model = Sequential()
3 model.add(Conv2D(32, kernel_size=(3, 3), #32 filtros de 3x3
4                 activation="relu",
5                 input_shape=input_shape))
6 model.add(MaxPooling2D(pool_size=(2, 2)))
7
8 model.add(Conv2D(64, kernel_size=(3, 3), activation="relu")) #64 filtros de 3x3
9 model.add(MaxPooling2D(pool_size=(2, 2)))
10
11 model.add(Flatten()) #Flatten para covertir el vector en uno unidimensional
12 model.add(Dropout(0.5)) #Apagamo el 50% de las neuronas para reforzar el aprendizaje
13 model.add(Dense(num_classes, activation="softmax"))
14 #esto hara que cada salida tenga un valor entre 0 y 1 de tipo probabilistico
15
16 model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010
Total params: 34,826		
Trainable params: 34,826		
Non-trainable params: 0		

## Compilación del modelo

```
1 #Compilamos el modelo
2 model.compile(loss="categorical_crossentropy",
3               optimizer="adam",
4               metrics=["accuracy"])
5
6 #Entrenamiento
7 entrenado = model.fit(train_images, train_labels,
8                       batch_size=batch_size,
9                       epochs=epochs,
10                      verbose=1)
```

```
Epoch 1/5
469/469 [=====] - 9s 18ms/step - loss: 0.3542 - accuracy: 0.8920
Epoch 2/5
469/469 [=====] - 8s 18ms/step - loss: 0.1016 - accuracy: 0.9687
Epoch 3/5
469/469 [=====] - 8s 18ms/step - loss: 0.0753 - accuracy: 0.9773
Epoch 4/5
469/469 [=====] - 8s 18ms/step - loss: 0.0655 - accuracy: 0.9797
Epoch 5/5
469/469 [=====] - 9s 19ms/step - loss: 0.0572 - accuracy: 0.9820
```

## Testeo del modelo entrenado

```
1 #Validation data
2 test_loss, test_accuracy = model.evaluate(test_images, test_labels, verbose=1)
3 print(f"TEST LOSS: {test_loss}")
4 print(f"TEST ACCURACY: {test_accuracy}")
5 print("")
6
7 #Historial
8 frame = pd.DataFrame(entrenado.history)
9 print("PRECISION POR EPOCAS")
10 print(frame)
```

```
313/313 [=====] - 1s 2ms/step - loss: 0.0325 - accuracy: 0.9891
TEST LOSS: 0.03251289203763008
TEST ACCURACY: 0.9890999794006348

PRECISION POR EPOCAS
      loss  accuracy
0  0.354207  0.892017
1  0.101618  0.968717
2  0.075293  0.977317
3  0.065530  0.979667
4  0.057199  0.981983
```

3)

Link al repo:

<https://github.com/LautaroLM/Machine-Learning-TP/tree/main/RandomForest>



## Importar librerías

```
In [40]: import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

%matplotlib inline
```

IMPORTAR  
LIBRERIAS

```
In [41]: data = pd.read_csv("diabetes.csv")
```

```
In [42]: data.shape
```

```
Out[42]: (768, 10)
```

```
In [43]: data.head()
```

```
Out[43]:
```

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	True
1	1	85	66	29	0	26.6	0.351	31	1.1426	False
2	8	183	64	0	0	23.3	0.672	32	0.0000	True
3	1	89	66	23	94	28.1	0.167	21	0.9062	False
4	0	137	40	35	168	43.1	2.288	33	1.3790	True

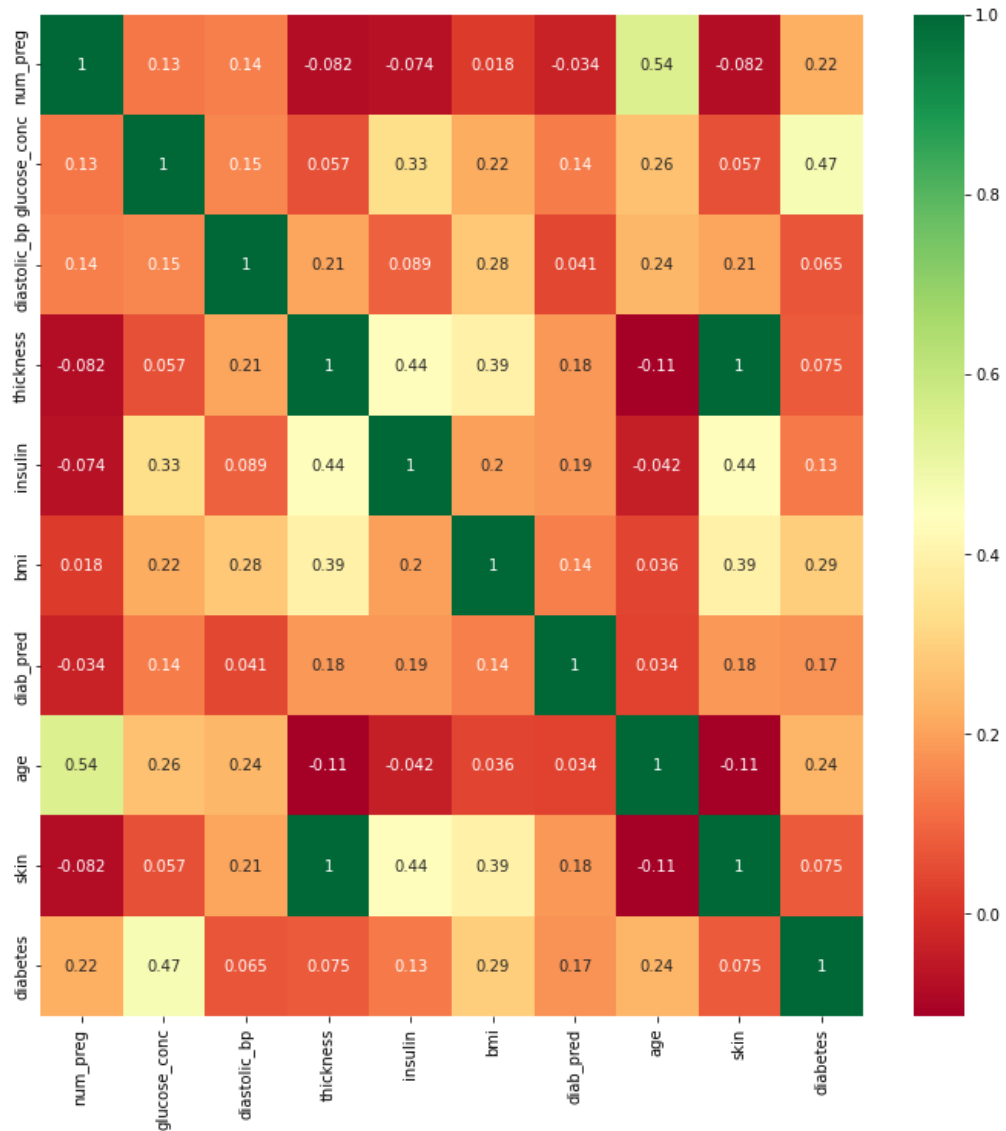
```
In [44]: # check if any null value is present
data.isnull().values.any()
```

INFO DE LA  
TABLA

```
Out[44]: False
```

```
In [45]: ## Graficar la correlación entre las columnas
import seaborn as sns
import matplotlib.pyplot as plt
corrmat = data.corr()
top_corr_features = corrmat.index
plt.figure(figsize=(12,12))
#plot heat map
g=sns.heatmap(data[top_corr_features].corr(),annot=True,cmap="RdYlGn")
```

Correlación entre los datos



```
In [47]: # Cambiamos la columna diabetes (true, false) por 1 ó 0.
diabetes_map = {True: 1, False: 0}
data['diabetes'] = data['diabetes'].map(diabetes_map)
```

```
In [48]: data.head()
```

```
Out[48]:
```

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	1
1	1	85	66	29	0	26.6	0.351	31	1.1426	0
2	8	183	64	0	0	23.3	0.672	32	0.0000	1
3	1	89	66	23	94	28.1	0.167	21	0.9062	0
4	0	137	40	35	168	43.1	2.288	33	1.3790	1

SE REEMPLAZAN  
LOS 'TRUE'  
'FALSE' POR 1 ó 0.

```
In [49]: # Imprimimos la cantidad de filas que tienen y no tienen diabetes. La suma debe dar 768.
diabetes_true_count = len(data.loc[data['diabetes'] == True])
diabetes_false_count = len(data.loc[data['diabetes'] == False])
(diabetes_true_count, diabetes_false_count)
```

```
Out[49]: (268, 500)
```

```
In [50]: ## Entrenamos el modelo.
```

```
from sklearn.model_selection import train_test_split
#feature_columns = ['num_preg', 'glucose_conc', 'diastolic_bp', 'insulin', 'bmi', 'diab_pred', 'age', 'skin']
feature_columns = ['num_preg', 'glucose_conc', 'bmi', 'age']
predicted_class = ['diabetes']
```

```
In [51]: X = data[feature_columns].values
y = data[predicted_class].values
```

ELEGIMOS LOS  
VALORES CON LOS  
QUE VAMOS A  
ENTRENAR

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.30, random_state=10)
```

Chequeamos que no haya valores en 0 en nuestro dataset. La cantidad de embarazos sí puede contener valores igual a 0.

```
In [52]: print("total number of rows : {}".format(len(data)))
print("number of rows missing glucose_conc: {}".format(len(data.loc[data['glucose_conc'] == 0])))
print("number of rows missing diastolic_bp: {}".format(len(data.loc[data['diastolic_bp'] == 0])))
print("number of rows missing insulin: {}".format(len(data.loc[data['insulin'] == 0])))
print("number of rows missing bmi: {}".format(len(data.loc[data['bmi'] == 0])))
print("number of rows missing diab_pred: {}".format(len(data.loc[data['diab_pred'] == 0])))
print("number of rows missing age: {}".format(len(data.loc[data['age'] == 0])))
print("number of rows missing skin: {}".format(len(data.loc[data['skin'] == 0])))
```

```
total number of rows : 768
number of rows missing glucose_conc: 5
number of rows missing diastolic_bp: 35
number of rows missing insulin: 374
number of rows missing bmi: 11
number of rows missing diab_pred: 0
number of rows missing age: 0
number of rows missing skin: 227
```

QUITAMOS LOS  
VALORES 0 Y  
ENTRENAMOS

```
In [53]: # Si dejamos esos valores en 0 nuestra predicción se va a ver afectada negativamente.
# Vamos a reemplazar esos valores en 0 a través de la estrategia 'mean'. 'mean' saca una media de los demás valores.
from sklearn.impute import SimpleImputer

fill_values = SimpleImputer(missing_values=0, strategy="mean")

X_train = fill_values.fit_transform(X_train)
X_test = fill_values.fit_transform(X_test)
```

```
In [55]: # Aplicamos el algoritmo RandomForest

from sklearn.ensemble import RandomForestClassifier
random_forest_model = RandomForestClassifier(random_state=10)

random_forest_model.fit(X_train, y_train.ravel())
```

SE APLICA  
ALGORITMO DE  
RANDOMFOREST

```
Out[55]: RandomForestClassifier(random_state=10)
```

```
In [56]: predict_train_data = random_forest_model.predict(X_test)

from sklearn import metrics

print("Accuracy = {:.3f}".format(metrics.accuracy_score(y_test, predict_train_data)))

Accuracy = 0.758
```

PRECISION DEL  
75%