Andrés Almécija

Table of Contents

Práctica No. 2. Preprocesado de datos.

Esta práctica constituye la segunda del módulo **Fundamentos de Machine Learning y Redes Neuronales** dentro del **Programa Executive en Artificial Intelligence** de **ThreePoints** dedicada al preprocesado de datos de acuerdo a lo especificado en la Unidad 2 del módulo.

Finalidad de la práctica

Dado el data set *Pima Indians Diabetes Database*, donde se tiene un registro de personas afectadas por Diabetes tipo 2 en función de otras muchas variables, se busca:

- **Familiarizarnos con las técnicas de importación** de datos a través de la librería `pandas` (https://pandas.pydata.org/).
- **Analizar** los datos haciendo uso de las librerías `pandas` (https://pandas.pydata.org/), `matplotlib` (https://matplotlib.org/) y `seaborn` (https://seaborn.pydata.org/).
- **Sacar la mayor cantidad de conclusiones posibles** de cara al posible entrenamiento de un modelo de Machine Learning
- **Realizar transformaciones y normalizaciones requeridas**
- **Mostrar un rápido ejemplo de regresión logística** con este dataset.

Característiscticas del dataset usado

Se hace uso del fichero `pima-indians-diabetes.csv` , el cual presenta las siguinetes características:

1. **Título**: Pima Indians Diabetes Database
2. **Fuentes**: (a) Original owners: National Institute of Diabetes and Digestive and

```
                    Kidney Diseases
```

   (b) Donor of database: Vincent Sigillito (vgs@aplcen.apl.jhu.edu)

```
                      Research Center, RMI Group Leader
                      Applied Physics Laboratory
                      The Johns Hopkins University
                      Johns Hopkins Road
                      Laurel, MD 20707
                      (301) 953-6231
```

   (c) Date received: 9 May 1990
3. **Referencias**:

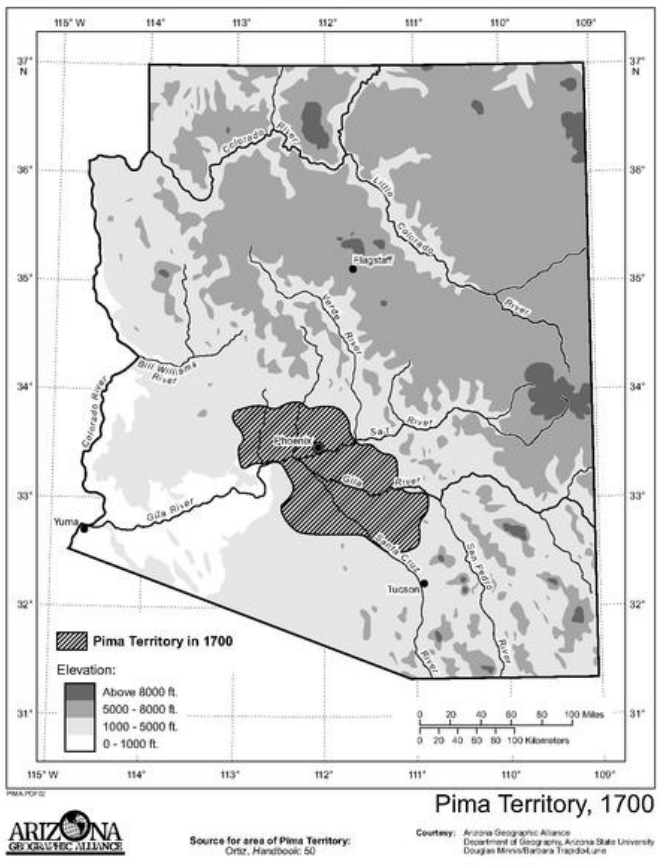
```
    Smith,~J.~W., Everhart,~J.~E., Dickson,~W.~C., Knowler,~W.~C., \&
    Johannes,~R.~S. (1988). Using the ADAP learning algorithm to forecast
    the onset of diabetes mellitus.  In {\it Proceedings of the Symposium
    on Computer Applications and Medical Care} (pp. 261--265).  IEEE
    Computer Society Press.

    The diagnostic, binary-valued variable investigated is whether the
    patient shows signs of diabetes according to World Health Organization
    criteria (i.e., if the 2 hour post-load plasma glucose was at least
    200 mg/dl at any survey  examination or if found during routine medical
    care).   The population lives near Phoenix, Arizona, USA.

    Results: Their ADAP algorithm makes a real-valued prediction between
    0 and 1.  This was transformed into a binary decision using a cutoff of
    0.448.  Using 576 training instances, the sensitivity and specificity
    of their algorithm was 76% on the remaining 192 instances.
```

4. **Información relevante**:

   Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. It is a unique algorithm; see the paper for details.
5. **Número de muestras**: 768
6. **Número de características**: 8 + clase
   A. Para cada característica: (**todas las variables son numéricas**)
      a. Number of times pregnant
      b. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
      c. Diastolic blood pressure (mm Hg)
      d. Triceps skin fold thickness (mm)
      e. 2-Hour serum insulin (mu U/ml)
      f. Body mass index (weight in kg/(height in m)^2)
      g. Diabetes pedigree function
      h. Age (years)
      i. Class variable (0 or 1)
7. **Missing Attribute Values**: Yes

Los **akimel o'odham** o **pima** son un grupo indígena que vive en el estado de Arizona (Estados Unidos) y en el estado mexicano en Sonora y Chihuahua. Su nombre significa "pueblo del río", que los distingue de sus parientes los "pápagos" (la gente del desierto). Se puede encontrar más informacién en [este enlace (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4418458/)](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4418458/).

In [216]:
```
from IPython.core.display import HTML
```



Pima Territory, 1700

Fotografías de miembros del pueblo Pima y Territorio ca. 1700

Importación del dataset

Comencemos importando las librerías que vamos a necesitar en esta parte de la práctica:

In [217]:
```
import pandas as pd
```

In [218]:
```
import numpy as np
```

Cargamos los datos desde el fichero `pima-indians-diabetes.csv` .

In [219]:
```
diabetes = pd.read_csv('data/pima-indians-diabetes.csv',
                       names=['Num_pregnant', 'Gluc_concent',
                              'Blood_press', 'Triceps', 'Insulin', 'BMI',
                              'Pedigree', 'Age', 'Diagnosis'])
```

In [220]:
```
diabetes.columns
```

Out[220]:
```
Index(['Num_pregnant', 'Gluc_concent', 'Blood_press', 'Triceps', 'Insulin',
       'BMI', 'Pedigree', 'Age', 'Diagnosis'],
      dtype='object')
```

Con la función `head` somos capaces de echarle un vistazo rápido a los valores del dataset.

In [221]:
```
diabetes.head(n=10)
```

Out[221]:

	Num_pregnant	Gluc_concent	Blood_press	Triceps	Insulin	BMI	Pedigree	Age	Diagnosis
0	6	148	72	35	0	33.60	0.63	50	1
1	1	85	66	29	0	26.60	0.35	31	0
2	8	183	64	0	0	23.30	0.67	32	1
3	1	89	66	23	94	28.10	0.17	21	0
4	0	137	40	35	168	43.10	2.29	33	1
5	5	116	74	0	0	25.60	0.20	30	0
6	3	78	50	32	88	31.00	0.25	26	1
7	10	115	0	0	0	35.30	0.13	29	0
8	2	197	70	45	543	30.50	0.16	53	1
9	8	125	96	0	0	0.00	0.23	54	1

Los tipos de cada columna se pueden explorar de la siguiente manera:

In [222]:
```
diabetes.dtypes
```

Out[222]:
```
Num_pregnant      int64
Gluc_concent      int64
Blood_press       int64
Triceps           int64
Insulin           int64
BMI             float64
Pedigree        float64
Age               int64
Diagnosis         int64
dtype: object
```

Primera pregunta (1 punto)

Por lo visto en el dataset anterior, para un caso de regresión logística en aprendizaje supervisado, ¿qué columnas son las variables que podríamos usar para realizar la clasificación y cuál es la columna que indica la etiqueta a la que se debe apuntar? Por favor, indíquelas.

Para realizar clasificación mediante regresión logística, las columnas que podríamos usar son las correspondientes a las siguientes variables: **Num_pregnant**,**Gluc_concent**,**Blood_press**,**Triceps**,**Insulin**,**BMI**,**Pedigree** y **Age**. La columna que indica la etiqueta es **Diagnosis**

Segunda pregunta (1 punto)

Haciendo uso de la función de pandas `isnull().sum()` , compruebe si hay `missing_values` en el dataset facilitado.

In [223]:
```
diabetes.info()
diabetes.isnull().sum()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Num_pregnant  768 non-null    int64  
 1   Gluc_concent  768 non-null    int64  
 2   Blood_press   768 non-null    int64  
 3   Triceps       768 non-null    int64  
 4   Insulin       768 non-null    int64  
 5   BMI           768 non-null    float64
 6   Pedigree      768 non-null    float64
 7   Age           768 non-null    int64  
 8   Diagnosis     768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Out[223]:
```
Num_pregnant    0
Gluc_concent    0
Blood_press     0
Triceps         0
Insulin         0
BMI             0
Pedigree        0
Age             0
Diagnosis       0
dtype: int64
```

El dataset tiene valores 0 que pueden considerarse como `missing_values` (por ejemplo la Blood_press 0 no es un valor válido), pero la función de pandas `isnull().sum()` no considera como `missing_values` a estos valores.

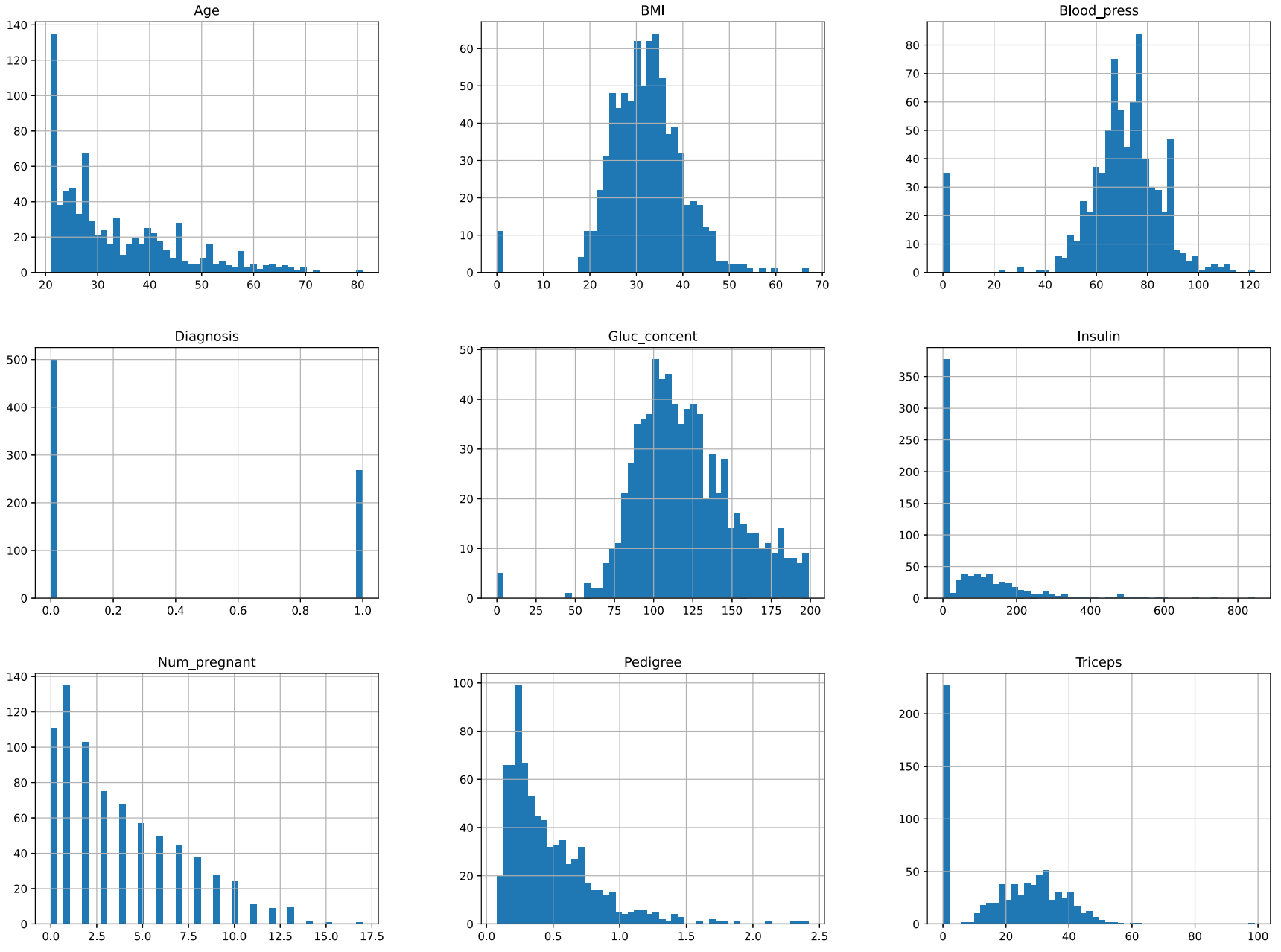
Visualización de datos

Tercera pregunta (2 puntos)

a) Haciendo uso de `df.hist()` , ejecute las siguientes líneas de código. (0.5 puntos)

Puede que resulte necesario corre dos veces la siguiente línea de código dado lo pesado del gráfico.

In [224]:
```
import matplotlib.pyplot as plt
diabetes.hist(bins = 50, figsize=(20, 15))
plt.show()
```



Se pueden obtener más detalles de nuestro dataset almacenados en `pandas` a través del comando `df.describe()`.

In [225]:
```
diabetes.describe()
```

Out[225]:

	Num_pregnant	Gluc_concent	Blood_press	Triceps	Insulin	BMI	Pedigree	Age	Diagnosis
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	120.89	69.11	20.54	79.80	31.99	0.47	33.24	0.35
std	3.37	31.97	19.36	15.95	115.24	7.88	0.33	11.76	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	0.08	21.00	0.00
25%	1.00	99.00	62.00	0.00	0.00	27.30	0.24	24.00	0.00
50%	3.00	117.00	72.00	23.00	30.50	32.00	0.37	29.00	0.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00	1.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00	1.00

b) Vemos que hay algunas características del dataset que tienen un valor mínimo sin ningún sentido lógico. ¿Qué características o variables considera que habría que modificar para eliminar esos registros nulos que carecen de sentido? (0.5 puntos)

Considero que habría que eliminar los valores 0 de las columnas Gluc_concent, Blood_press, Triceps, Insulin y BMI ya que no tienen sentido.

c) ¿Cuál es la relación $\Gamma = \frac{diabeticos}{\neg diabeticos}$? ¿Está nuestro dataset balanceado? Si es así, justificar. De lo contrario, indicar qué clase está subrepresentada. (0.5 puntos)

In [226]:
```
diabetes['Diagnosis'].value_counts()
```

Out[226]:
```
0    500
1    268
Name: Diagnosis, dtype: int64
```

El dataset no está balanceado, debido a que hay 500 ejemplos de clasificaciones 0 (no diabetes) y 268 clasificaciones 1 (si diabetes). Entonces, la clase sub representada es "1 (si diabetes)"

d) ¿Qué alternativas de actuación sobre el dataset propone para compensar la posible subrepresentación de una clase? (0.5 puntos)

Existen técnicas para compensar la subrepresentación de una clase.

Por ejemplo:

- **subsampling** que consiste en reducir la clase mayoritaria.
- **oversampling** que consiste en crear nuevas muestras en la clase minoritaria.
- Combinación de **subsampling** y **oversampling**.

Cada una de ellas con sus ventajas y desventajas

Una librería muy popular que permite aplicar estas técnicas es `imblearn` (https://imbalanced-learn.readthedocs.io/en/stable/ (https://imbalanced-learn.readthedocs.io/en/stable/))

Limpieza y procesado de datos

Los algoritmos de Machine Learning no suelen funcionar muy bien cuando faltan datos en los mismos, ya sea por la presencia de `missing values` o por la presencia de $0s$, por lo que debemos encontrar una solución para mitigar estos efectos.

Cuarta pregunta (1 puntos)

¿Cuál es el porcentaje de datos nulos de insulina en el dataset (en la columna `insuline`)? ¿Ve conveniente eliminar todas aquellas filas donde nos falte alguna variable? Justifique su respuesta.

In [227]:
```
print(diabetes[diabetes['Insulin'] == 0].count()[0] / diabetes['Insulin'].count() * 100, "%")
```

```
48.69791666666667 %
```

No veo conveniente eliminar las filas ya que se descartaría información valiosa. Además serían demasiados los registros eliminados.

Una opción reside en calcular el valor mediano para una columna específica y sustituir ese valor en todas partes (en la misma columna) donde tenemos cero o nulo. (Pequeña ayuda para sustituir los ceros por la mediana de una columna)

In [228]:
```
median_var = diabetes['Insulin'].median()
# Sustituimos los valores nulos del índice de una determinada variable por la mediana
diabetes['Insulin'] = diabetes['Insulin'].replace(to_replace = 0, value = median_var)
```

In [229]:
```
diabetes['Insulin'].describe()
```

Out[229]:
```
count   768.00
mean     94.65
std     105.55
min      14.00
25%      30.50
50%      31.25
75%     127.25
max     846.00
Name: Insulin, dtype: float64
```

Quinta pregunta (1 puntos)

a) Realice la sustitución indicada arriba por la mediana en todas aquellas variables que considere que contienen valores sin sentido (en relación con la pregunta 3.b.) (5 puntos)

In [230]:
```
median_var = diabetes['Gluc_concent'].median()
diabetes['Gluc_concent'] = diabetes['Gluc_concent'].replace(to_replace = 0, value = median_var)

median_var = diabetes['Blood_press'].median()
diabetes['Blood_press'] = diabetes['Blood_press'].replace(to_replace = 0, value = median_var)

median_var = diabetes['Triceps'].median()
diabetes['Triceps'] = diabetes['Triceps'].replace(to_replace = 0, value = median_var)

#median_var = diabetes['Insulin'].median()
#diabetes['Insulin'] = diabetes['Insulin'].replace(to_replace = 0, value = median_var)

median_var = diabetes['BMI'].median()
diabetes['BMI'] = diabetes['BMI'].replace(to_replace = 0, value = median_var)
```

b) Vuelva a ejecutar el comando `diabetes.describe()`. ¿Cuál es la única variable que debe tener como valor mínimo 0 tras haber hecho la sustitución de los valores nulos por su mediana? (0,5 puntos)

In [231]:
```
diabetes.describe()
```

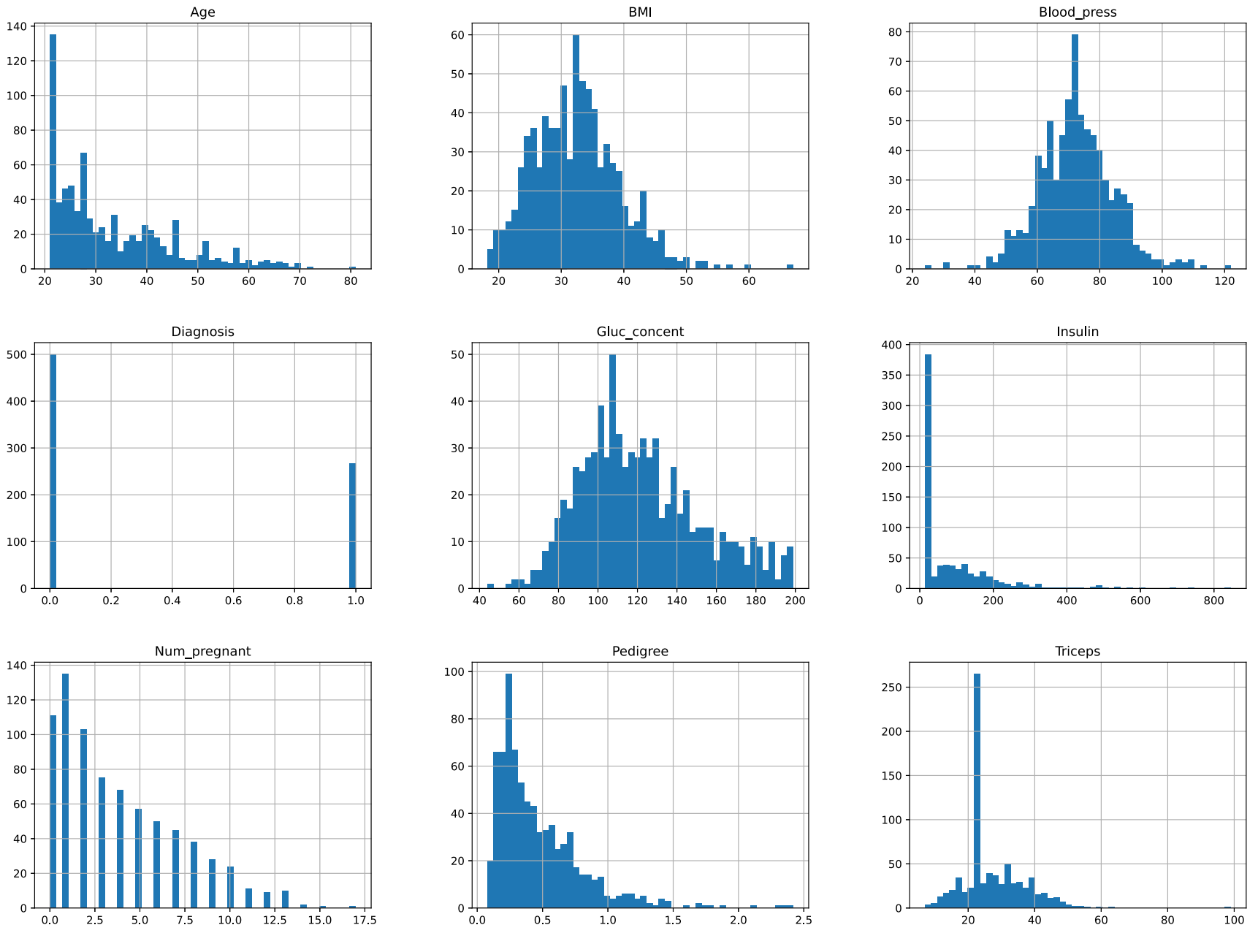
Out[231]:

	Num_pregnant	Gluc_concent	Blood_press	Triceps	Insulin	BMI	Pedigree	Age	Diagnosis
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	3.85	121.66	72.39	27.33	94.65	32.45	0.47	33.24	0.35
std	3.37	30.44	12.10	9.23	105.55	6.88	0.33	11.76	0.48
min	0.00	44.00	24.00	7.00	14.00	18.20	0.08	21.00	0.00
25%	1.00	99.75	64.00	23.00	30.50	27.50	0.24	24.00	0.00
50%	3.00	117.00	72.00	23.00	31.25	32.00	0.37	29.00	0.00
75%	6.00	140.25	80.00	32.00	127.25	36.60	0.63	41.00	1.00
max	17.00	199.00	122.00	99.00	846.00	67.10	2.42	81.00	1.00

Tras haber hecho la sustitución de los valores nulos por su mediana, la única columna que tiene sentido que tenga un valor mínimo 0 es **Num_pregnant**

Vuelva a ejecutar el siguiento comando para la visualización de datos:

In [232]:
```
import matplotlib.pyplot as plt
diabetes.hist(bins = 50, figsize=(20, 15))
plt.show()
```



Escalado del dataset

Normalizar el dataset para su uso en ciertos algoritmos de Machine Learning se convierte en tarea imprescindible

Sexta pregunta (2 puntos)

a) Escale las caracteríticas del dataset que considere necesarias entre `0` y `1` para su posterior uso en una regresión logítica binaria. (1 puntos)

Pista: para hacerlo rápidamente, puede definir una lista `cols_to_norm` con las columnas a normalizar e incluir la función a usar en el escalado e iterar en bucle:

In [233]:
```
cols_to_norm = ['Num_pregnant', 'Gluc_concent', 'Blood_press', 'Triceps', 'Insulin', 'BMI', 'Age', 'Pedigree']
```

cols_to_norm = [...]

In [234]:
```
for item in cols_to_norm:
    diabetes[item]= (diabetes[item] - np.min(diabetes[item]))/(np.max(diabetes[item]) - np.min(diabetes[item]))
```

Ejecute de nuevo los siguientes comandos:

In [235]:
```
diabetes.head(n=10)
```

Out[235]:

	Num_pregnant	Gluc_concent	Blood_press	Triceps	Insulin	BMI	Pedigree	Age	Diagnosis
0	0.35	0.67	0.49	0.30	0.02	0.31	0.23	0.48	1
1	0.06	0.26	0.43	0.24	0.02	0.17	0.12	0.17	0
2	0.47	0.90	0.41	0.17	0.02	0.10	0.25	0.18	1
3	0.06	0.29	0.43	0.17	0.10	0.20	0.04	0.00	0
4	0.00	0.60	0.16	0.30	0.19	0.51	0.94	0.20	1
5	0.29	0.46	0.51	0.17	0.02	0.15	0.05	0.15	0
6	0.18	0.22	0.27	0.27	0.09	0.26	0.07	0.08	1
7	0.59	0.46	0.49	0.17	0.02	0.35	0.02	0.13	0
8	0.12	0.99	0.47	0.41	0.64	0.25	0.03	0.53	1
9	0.47	0.52	0.73	0.17	0.02	0.28	0.07	0.55	1

In [236]:
```
diabetes.describe()
```

Out[236]:

	Num_pregnant	Gluc_concent	Blood_press	Triceps	Insulin	BMI	Pedigree	Age	Diagnosis
count	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00	768.00
mean	0.23	0.50	0.49	0.22	0.10	0.29	0.17	0.20	0.35
std	0.20	0.20	0.12	0.10	0.13	0.14	0.14	0.20	0.48
min	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
25%	0.06	0.36	0.41	0.17	0.02	0.19	0.07	0.05	0.00
50%	0.18	0.47	0.49	0.17	0.02	0.28	0.13	0.13	0.00
75%	0.35	0.62	0.57	0.27	0.14	0.38	0.23	0.33	1.00
max	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

b) ¿Qué destacaría en las dos tablas anteriores?¿Cómo han cambiado los intervalos en los que las características están distribuidas? (0,5 puntos)

Una vez aplicado el escalamiento, todas las columnas varían entre 0 y 1. En la segunda tabla se ve que para todas las columnas el valor mínimo es 0 y el valor máximo es 1.

c) ¿Qué otro tipo de normalización aplicaría a las características de nuestro dataset? Impleméntela debajo y justifique su elección. (0,5 puntos)

In [237]:
```
#Se podría normalizar el dataset de manera que cada en caracterísitica la media sea 0 y la desviación estandar 1

#for item in cols_to_norm:
#    diabetes[item]= (diabetes[item] - diabetes[item].mean())/diabetes[item].std()
```

Matriz de correlación

En `pandas`, es muy sencillo obtener la visualización de la matriz de correlación. **Ejecute el siguiente código:**

In [238]:
```
corr = diabetes.corr()
corr
```

Out[238]:

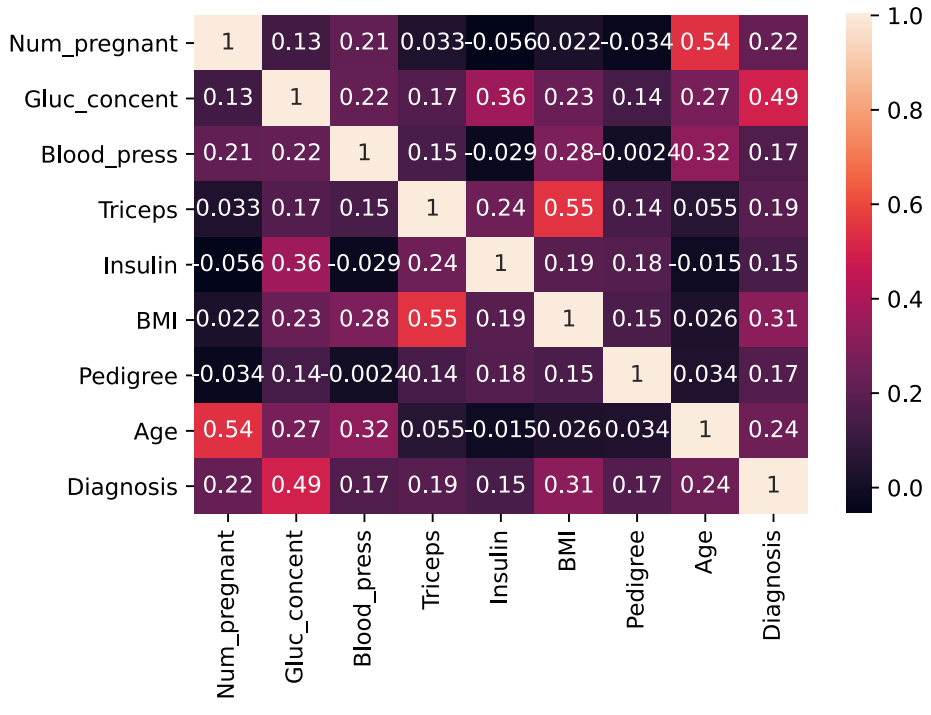
	Num_pregnant	Gluc_concent	Blood_press	Triceps	Insulin	BMI	Pedigree	Age	Diagnosis
Num_pregnant	1.00	0.13	0.21	0.03	-0.06	0.02	-0.03	0.54	0.22
Gluc_concent	0.13	1.00	0.22	0.17	0.36	0.23	0.14	0.27	0.49
Blood_press	0.21	0.22	1.00	0.15	-0.03	0.28	-0.00	0.32	0.17
Triceps	0.03	0.17	0.15	1.00	0.24	0.55	0.14	0.05	0.19
Insulin	-0.06	0.36	-0.03	0.24	1.00	0.19	0.18	-0.02	0.15
BMI	0.02	0.23	0.28	0.55	0.19	1.00	0.15	0.03	0.31
Pedigree	-0.03	0.14	-0.00	0.14	0.18	0.15	1.00	0.03	0.17
Age	0.54	0.27	0.32	0.05	-0.02	0.03	0.03	1.00	0.24
Diagnosis	0.22	0.49	0.17	0.19	0.15	0.31	0.17	0.24	1.00

Haciendo uso de `seaborn` podremos hacernos una idea rápida de qué variables están más correlacionadas con la clase. Ejecute el siguiente código.

In [239]:
```
%matplotlib inline
import seaborn as sns
```

In [240]:
```
sns.heatmap(corr, annot = True)
```

Out[240]: <matplotlib.axes._subplots.AxesSubplot at 0x1dfd3d60>



Séptima pregunta (1 punto)

Indique, de mayor a menor correlación, las variables que guardan una mayor correlación con la clase. ¿Tiene sentido que las tres primeras variables con mayor correlación se identifiquen con una mayor probabilidad de sufrir diabetes de tipo 2? Justifique su respuesta.

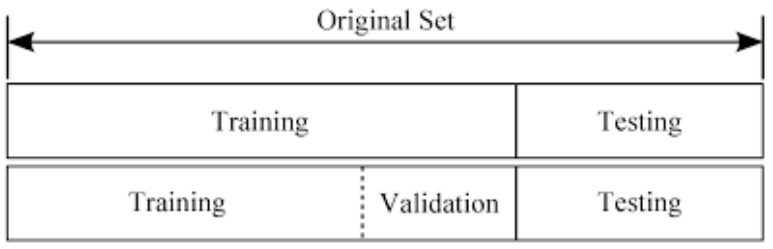
Las variables que presentan una mayor correlación con la clase son ´Gluc_concent´, ´BMI´ y ´Age´. Sin mucho conocimiento médico, me atrevo a decir en base a lo expuesto por la matriz de correlacion, que la concentración de glucosa, el BMI y la edad tienen influencia en que una persona sufra de diabetes.

Entrenando un modelo de Regresión Logística para el dataset

Ahora podemos entrenar un modelo de clasificación. Usaremos un modelo simple de aprendizaje automático llamado Regresión Logística. Dado que el modelo está disponible en Scikit-sklearn, el proceso de capacitación es bastante sencillo y podemos hacerlo en pocas líneas de código. Primero, creamos una instancia llamada `diabetesCheck` y luego usamos la función de ajuste para entrenar el modelo.

Partición del dataset en train test y test set

Antes de comenzar el entrenamiento de nuestro modelo, necesitamos hacer una **partición de nuestro datos** entre aquellos que se van a usar para entrenar y aquellos que van a usarse para la validación de ese mismo entrenamiento. Esa partición suele ser aleatoria, y con `train_test_split` podremos hacer la partición fácilmente:



Por favor, **ejecute las siguientes líneas de código**:

In [241]:
```
x_data = diabetes.drop('Diagnosis', axis=1)
```

In [242]:
```
x_data.shape
```

Out[242]: (768, 8)

In [243]:
```
labels = diabetes['Diagnosis']
```

In [244]:
```
labels.shape
```

Out[244]: (768,)

Con esto ya hemos aislado las características de las etiquetas desde el `pandas dataframe` en el que estábamos trabajando.

Importamos `train_test_split` y hacemos la partición de nuestros datos

In [245]:
```
from sklearn.model_selection import train_test_split
```

In [246]:
```
X_train, X_test, y_train, y_test = train_test_split(x_data, labels, test_size=0.33, random_state=101)
```

Importamos el constructor `LogisticRegression` para **comenzar con la regresión logística** y lanzamos el entrenamiento con `diabetesCheck.fit()` sobre el dataset de entrenamiento.

In [247]:
```
from sklearn.linear_model import LogisticRegression
```

In [248]:
```
diabetesCheck = LogisticRegression(solver='liblinear')
```

In [249]:
```
diabetesCheck.fit(X_train, y_train)
```

Out[249]:
```
LogisticRegression(solver='liblinear')
```

Para obtener la precisión de nuestro modelo, podemos atacar directamente a nuestro `test dataset` con `diabetesCheck.score()` .

In [250]:
```
accuracy = diabetesCheck.score(X_test, y_test)
print("accuracy =", accuracy * 100, "%")
```

```
accuracy = 76.37795275590551 %
```

Esta es la precisión de nuestro modelo tras habér realizado una Regresión Logística.

Octava pregunta (1 punto)

Este ajuste se ha realizado con los datos escalados entre 0 y 1.

a) Repita este ajuste con los datos no escalados (es decir, con el dataset original) (0,4 puntos)

¡Necesitará cargar el dataset desde el fichero original!

In [251]:
```
#Cargar el dataset orginal
diabetes_original = pd.read_csv('data/pima-indians-diabetes.csv',
                         names=['Num_pregnant', 'Gluc_concent',
                                'Blood_press', 'Triceps', 'Insulin', 'BMI',
                                'Pedigree', 'Age', 'Diagnosis'])

#Eliminar los 0s, reemplazando por la media
cols_to_reeplace = ['Gluc_concent', 'Blood_press', 'Triceps', 'Insulin', 'BMI', 'Age', 'Pedigree']
for item in cols_to_reeplace:
    median_var = diabetes_original[item].median()
    diabetes_original[item] = diabetes_original[item].replace(to_replace = 0, value = median_var)

#Separar el dataset
x_data_original = diabetes_original.drop('Diagnosis', axis=1)
labels_original = diabetes_original['Diagnosis']
X_train_original, X_test_original, y_train_original, y_test_original = train_test_split(x_data_original, labels_original, test_siz
e=0.33, random_state=101)

#Entrenar
diabetesCheckOriginal = LogisticRegression(solver='liblinear')
diabetesCheckOriginal.fit(X_train_original, y_train_original)

#Evaluar
accuracy_original = diabetesCheckOriginal.score(X_test_original, y_test_original)
print("accuracy original=", accuracy_original * 100, "%")
```

```
accuracy original= 74.01574803149606 %
```

b) Repita este ajuste con los datos normalizados con $\mu = 0$ y $\sigma = 1$. (0,3 puntos)

¡Necesitará cargar el dataset desde el fichero original!

In [252]:
```
#Cargar el dataset orginal
pd.options.display.float_format = '{:,.2f}'.format
diabetes_norm = pd.read_csv('data/pima-indians-diabetes.csv',
                        names=['Num_pregnant', 'Gluc_concent',
                               'Blood_press', 'Triceps', 'Insulin', 'BMI',
                               'Pedigree', 'Age', 'Diagnosis'])

#Eliminar los 0s, reemplazando por la media
cols_to_reeplace = ['Gluc_concent', 'Blood_press', 'Triceps', 'Insulin', 'BMI', 'Age', 'Pedigree']

for item in cols_to_reeplace:
    median_var = diabetes_norm[item].median()
    diabetes_norm[item] = diabetes_norm[item].replace(to_replace = 0, value = median_var)

#Normalizar
cols_to_norm = ['Num_pregnant', 'Gluc_concent', 'Blood_press', 'Triceps', 'Insulin', 'BMI', 'Age', 'Pedigree']
for item in cols_to_norm:
    diabetes_norm[item]= (diabetes_norm[item] - diabetes_norm[item].mean())/diabetes_norm[item].std()

#Separar el dataset
x_data_norm = diabetes_norm.drop('Diagnosis', axis=1)
labels_norm = diabetes_norm['Diagnosis']
X_train_norm, X_test_norm, y_train_norm, y_test_norm = train_test_split(x_data_norm, labels_norm, test_size=0.33, random_state=101
)

#Verificar mu y sigma
print(X_train_norm.mean())
print(X_train_norm.std())

#Entrenar
diabetesCheckNorm = LogisticRegression(solver='liblinear')
diabetesCheckNorm.fit(X_train_norm, y_train_norm)

#Evaluar
accuracy_norm = diabetesCheckNorm.score(X_test_norm, y_test_norm)
print("accuracy norm=", accuracy_norm * 100, "%")
```

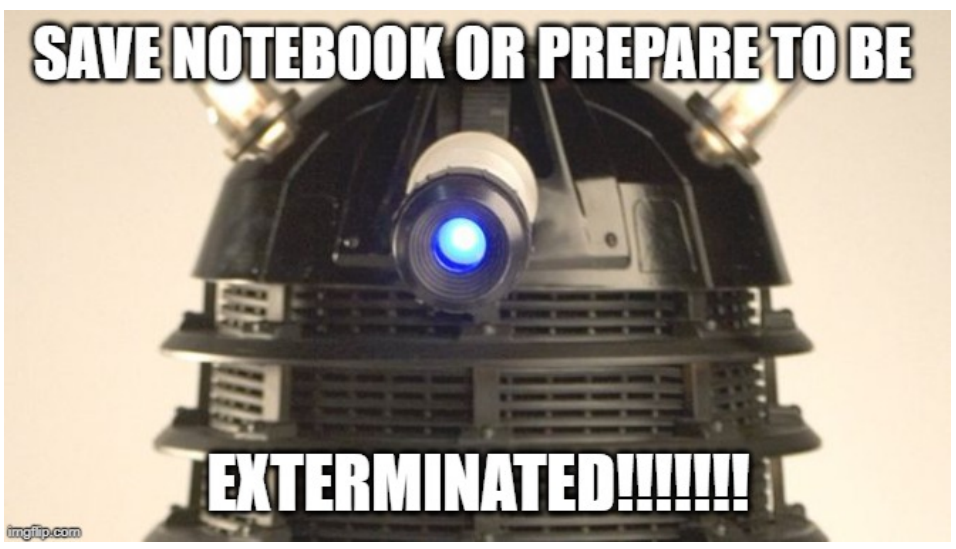
```
Num_pregnant   -0.03
Gluc_concent    0.01
Blood_press     0.00
Triceps        -0.00
Insulin         0.02
BMI             0.02
Pedigree        0.00
Age            -0.00
dtype: float64
Num_pregnant   0.98
Gluc_concent   0.98
Blood_press    0.97
Triceps        1.03
Insulin        0.99
BMI            1.01
Pedigree       0.93
Age            1.02
dtype: float64
accuracy norm= 77.55905511811024 %
```

c) ¿Observa una mejora sustancial en alguno de los casos o un empeoramiento? Justifique su respuesta. (0,3 puntos)

In [253]:
```
print("accuracy escalado=", accuracy * 100, "%")
print("accuracy original=", accuracy_original * 100, "%")
print("accuracy normalizado=", accuracy_norm * 100, "%")
```

```
accuracy escalado= 76.37795275590551 %
accuracy original= 74.01574803149606 %
accuracy normalizado= 77.55905511811024 %
```

Se ve una leve mejora escalando o normalizando.
- Con respecto al escalado, debido a que el orden de magnitud de las variables es similar, al unificar la escala hay mejora, pero no es sustancial
- Respecto a la normalización, ocurre algo similar al escalado: la mayoría de las caracterísiticas parecen responder a una distribución normal y por eso no se gana en exceso al estandarizar a media 0 y sigma 1.



¡Por favor, no olvide guardar el Jupyter Notebook antes de mandar la práctica!

In [ ]: