# 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**

**preprocesado de datos de acuerdo a lo especificado en la Unidad 2 del modulo.**

# 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`.
- **Analizar los datos haciendo uso de las librerías** `pandas`, `matplotlib` **y** `seaborn`.
- **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.**

---

# Característstiscticas 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:**

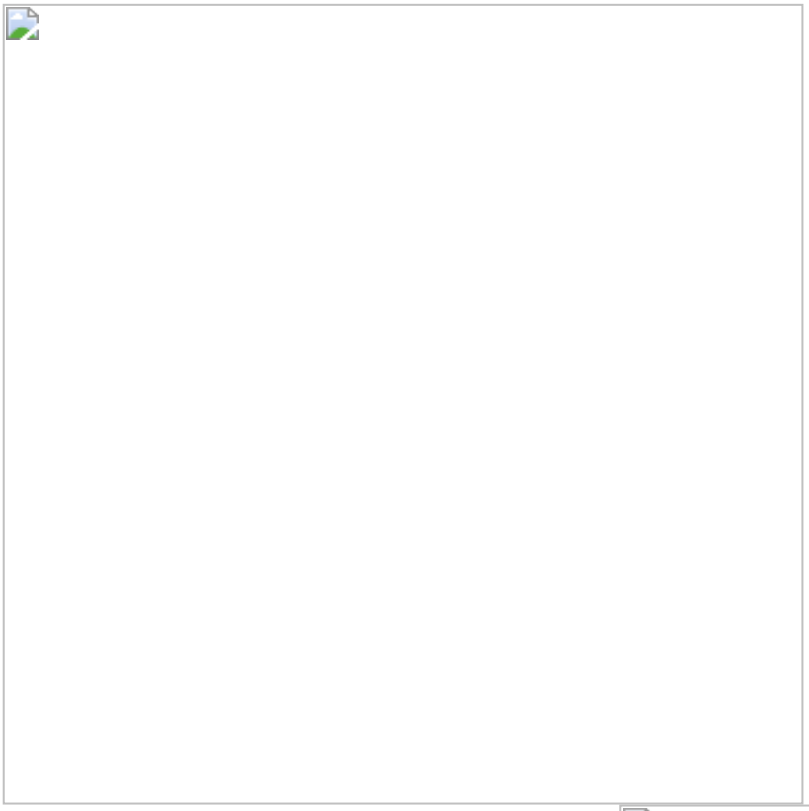
   **Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. ADAP is an adaptive learning routine that generates and executes digital analogs of perceptron-like devices. It is a unique algorithm; see the paper for details.**

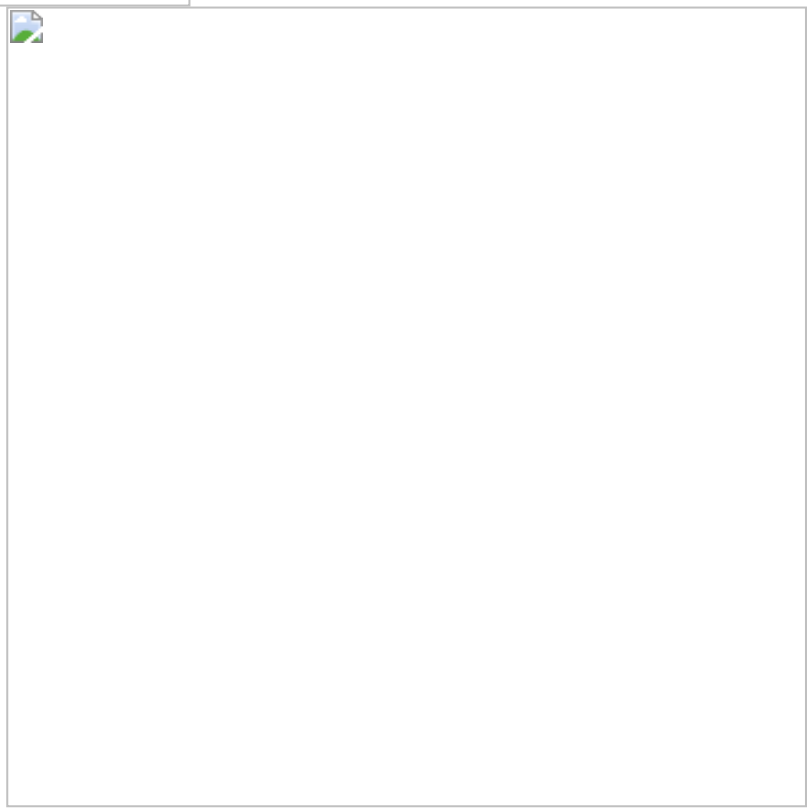
5. **Número de muestras: 768**
6. **Número de características: 8 + clase**
7. **Para cada característica: (todas las variables son numéricas)**
   1. **Number of times pregnant**
   2. **Plasma glucose concentration a 2 hours in an oral glucose tolerance test**
   3. **Diastolic blood pressure (mm Hg)**
   4. **Triceps skin fold thickness (mm)**
   5. **2-Hour serum insulin (mu U/ml)**
   6. **Body mass index (weight in kg/(height in m)^2)**
   7. **Diabetes pedigree function**
   8. **Age (years)**
   9. **Class variable (0 or 1)**
8. **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**

**los distingue de sus parientes los "papagos" (la gente del desierto). Se puede encontrar más información en [este enlace](#).**

```
from IPython.core.display import HTML
```





**Tipo de celda no admitido. Haz doble clic para inspeccionar/editar el contenido.**

---

# Importación del dataset

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

```
import pandas as pd
```

```
import numpy as np
```

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

```
diabetes = pd.read_csv('https://raw.githubusercontent.com/wilberj88/Diabetes/master/pi
                       names=['Num_pregnant', 'Gluc_concent',
                              'Blood_press', 'Triceps', 'Insulin', 'BMI',
                              'Pedigree', 'Age', 'Diagnosis'])
```

```
diabetes.columns
```

```
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.

```
diabetes.head(n=10)
```

| | Num_pregnant | Gluc_concent | Blood_press | Triceps | Insulin | BMI | Pedigree | Age |
|---|---|---|---|---|---|---|---|---|
| 0 | 6 | 148 | 72 | 35 | 0 | 33.6 | 0.627 | 50 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 85 | 66 | 29 | 0 | 26.6 | 0.351 | 31 |

## 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.**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | 2 | 197 | 70 | 45 | 543 | 30.5 | 0.158 | 53 |

La etiqueta a la que debe apuntar es a la de diagnosis como variable dependiente y con

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

```
diabetes.dtypes
```

```
Num_pregnant      int64
Gluc_concent      int64
Blood_press       int64
Triceps           int64
Insulin           int64
BMI             float64
Pedigree        float64
Age               int64
Diagnosis         int64
dtype: object
```

## Segunda pregunta (1 punto)

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

```
diabetes.sum()
```

```
Num_pregnant     2953.000
Gluc_concent    92847.000
Blood_press     53073.000
```

```
Triceps          15772.000
```

```
diabetes.isnull().sum()
```

```
Num_pregnant    0
Gluc_concent    0
Blood_press     0
Triceps         0
Insulin         0
BMI             0
Pedigree        0
Age             0
Diagnosis       0
dtype: int64
```

```
En este dataset no hay missing_values
```

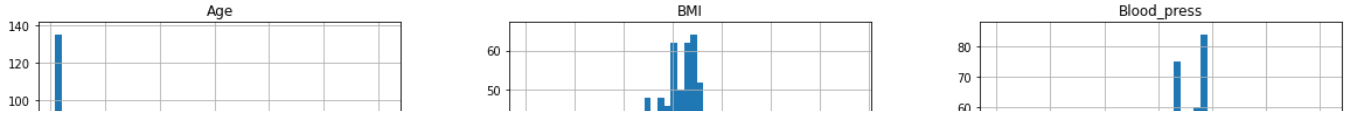
## Visualización de datos

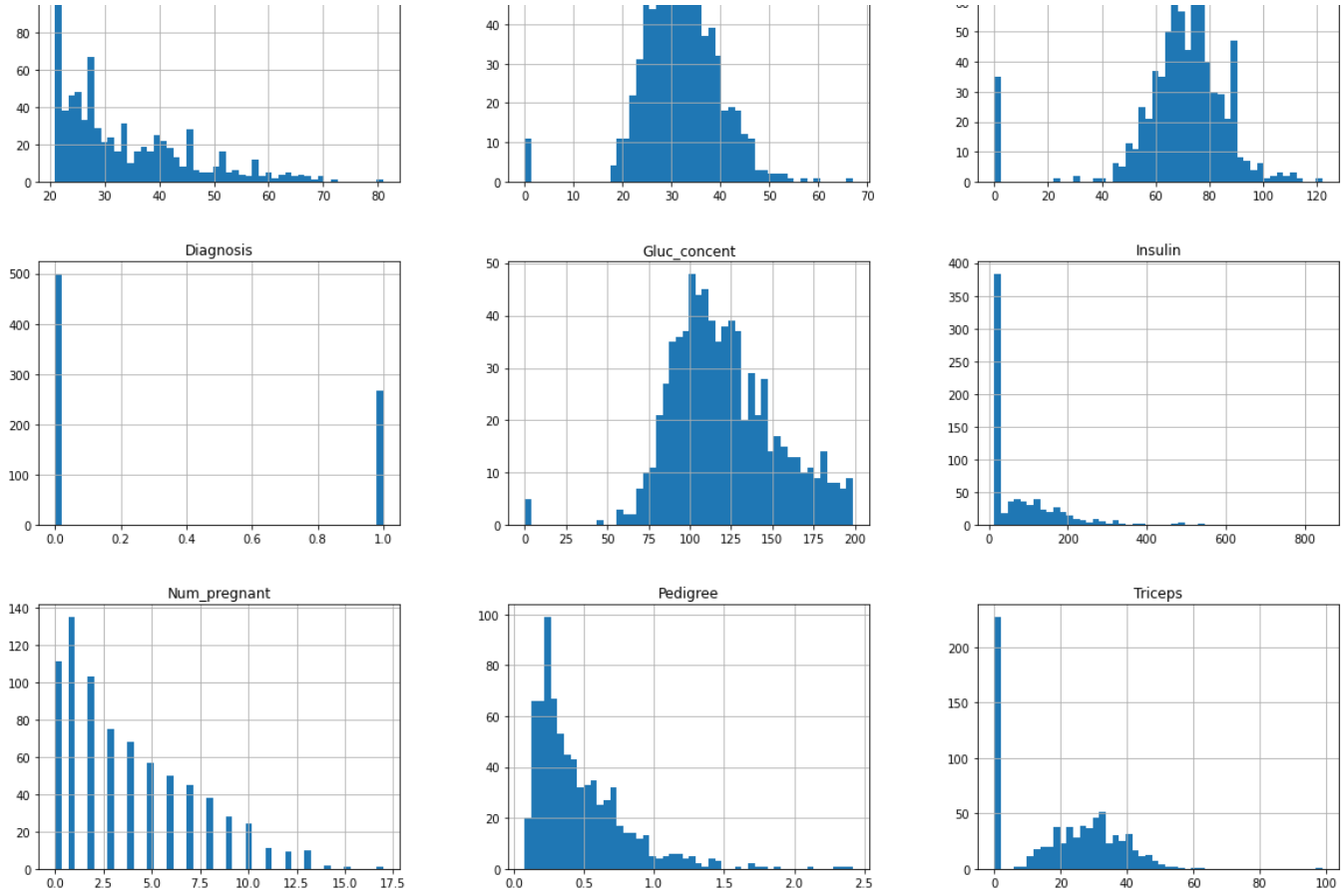
## Tercera pregunta (2 puntos)

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

**Tipo de celda no admitido. Haz doble clic para inspeccionar/editar el contenido.**

```
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()`.**

```
diabetes.describe()
```

| | Num_pregnant | Gluc_concent | Blood_press | Triceps | Insulin | BMI | |
|---|---|---|---|---|---|---|---|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 7 |

### 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)

```
La variable de presión sanguínea, pues si tiene cero de presión está muerto el pacient
```

### 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)

```
Sí, se encuentra balanceado el dataset toda vez que de 768 muestras, 2/3 son negativos
```

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

```
Hacer composiciones del dataset balanceadas al 50% entre las muestras con la etiqueta
```

---

## 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.**

**Tipo de celda no admitido. Haz doble clic para inspeccionar/editar el contenido.**

### 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

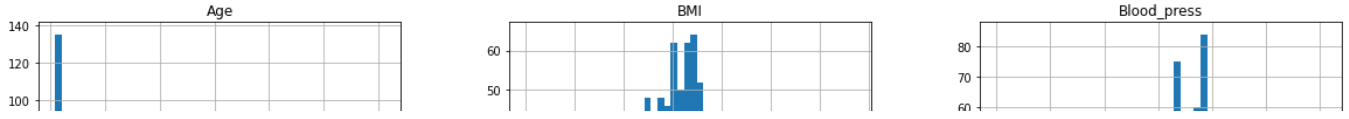
## alguna variable? Justifique su respuesta.

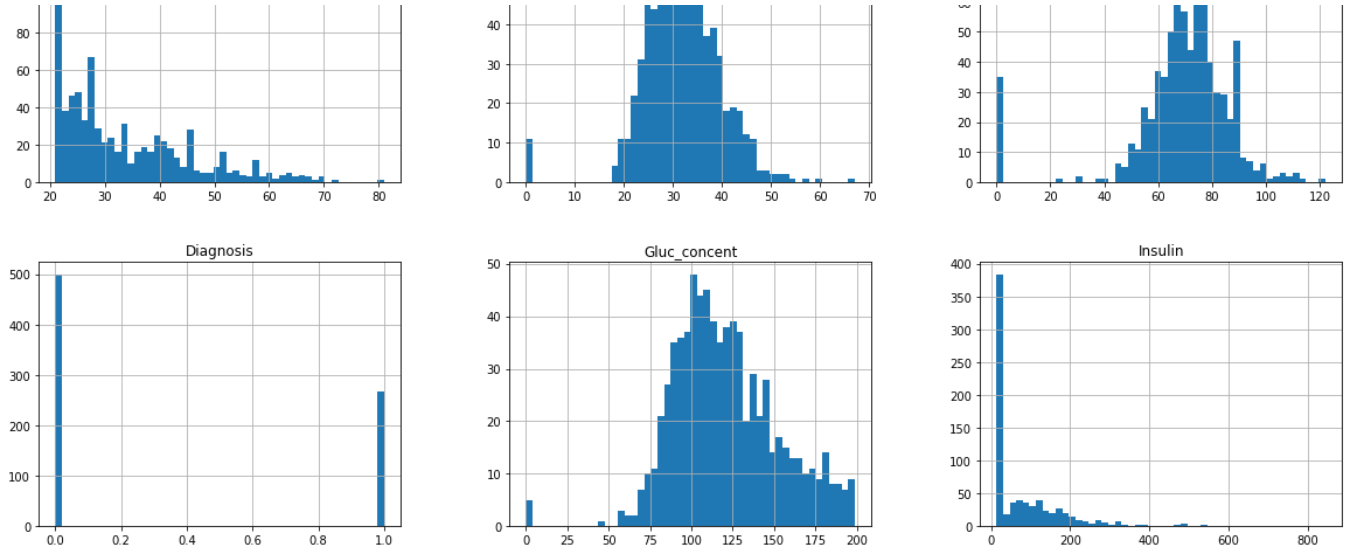
El porcentaje de datos nulos de insulina en el dataset es de 0%. Para este dataset no

### 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)

```
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)
```

```
import matplotlib.pyplot as plt
diabetes.hist(bins = 50, figsize=(20, 15))
plt.show()
```






## 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)

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

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

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

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)

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

**Haz doble clic (o pulsa Intro) para editar**

### 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

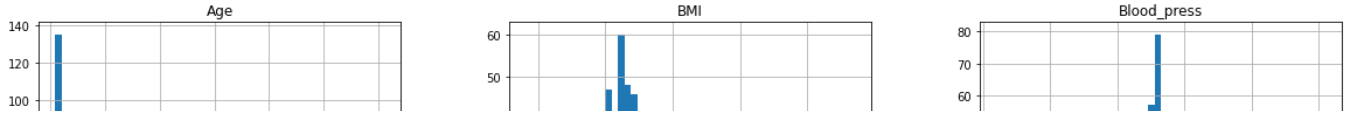
## de los valores nulos por su mediana? (0,5 puntos)

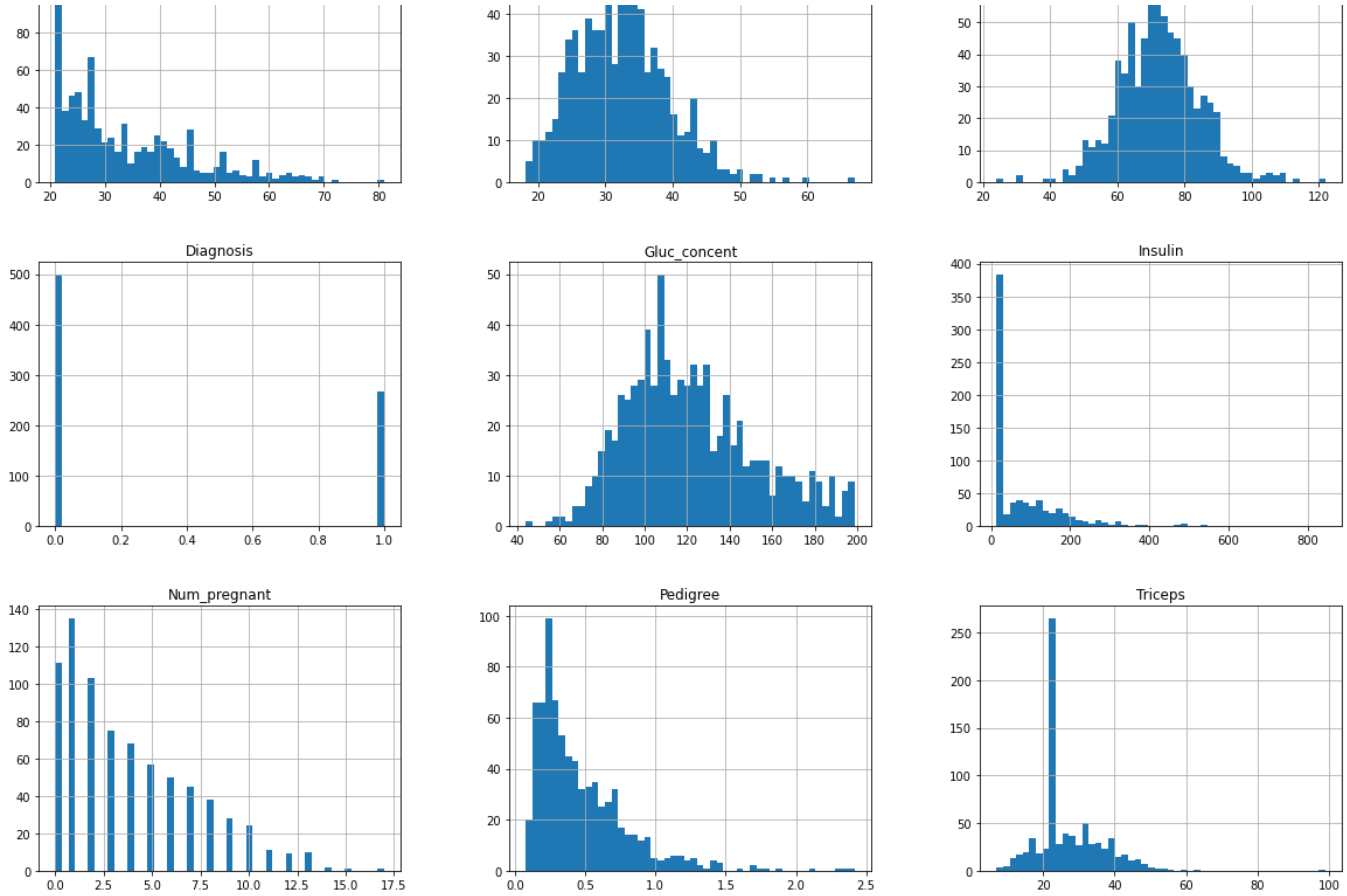
```
diabetes.describe()
```

| | Num_pregnant | Gluc_concent | Blood_press | Triceps | Insulin | BMI | 7 |
|---|---|---|---|---|---|---|---|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | |
| mean | 3.845052 | 121.656250 | 72.386719 | 27.334635 | 94.652344 | 32.450911 | |
| std | 3.369578 | 30.438286 | 12.096642 | 9.229014 | 105.547598 | 6.875366 | |
| min | 0.000000 | 44.000000 | 24.000000 | 7.000000 | 14.000000 | 18.200000 | |
| 25% | 1.000000 | 99.750000 | 64.000000 | 23.000000 | 30.500000 | 27.500000 | |
| 50% | 3.000000 | 117.000000 | 72.000000 | 23.000000 | 31.250000 | 32.000000 | |
| 75% | 6.000000 | 140.250000 | 80.000000 | 32.000000 | 127.250000 | 36.600000 | |
| max | 17.000000 | 199.000000 | 122.000000 | 99.000000 | 846.000000 | 67.100000 | |

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

```
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 convie

### 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:

```
cols_to_norm = ['Num_pregnant', 'Gluc_concent', 'Blood_press', 'Triceps', 'Insulin',
```

**Tipo de celda no admitido. Haz doble clic para inspeccionar/editar el contenido.**

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

**Ejecute de nuevo los siguientes comandos:**

```
diabetes.head(n=10)
```

| | Num_pregnant | Gluc_concent | Blood_press | Triceps | Insulin | BMI | Pedigree |
|---|---|---|---|---|---|---|---|
| 0 | 0.352941 | 0.670968 | 0.489796 | 0.304348 | 0.019832 | 0.314928 | 0.234415 |
| 1 | 0.058824 | 0.264516 | 0.428571 | 0.239130 | 0.019832 | 0.171779 | 0.116567 |
| 2 | 0.470588 | 0.896774 | 0.408163 | 0.173913 | 0.019832 | 0.104294 | 0.253629 |
| 3 | 0.058824 | 0.290323 | 0.428571 | 0.173913 | 0.096154 | 0.202454 | 0.038002 |
| 4 | 0.000000 | 0.600000 | 0.163265 | 0.304348 | 0.185096 | 0.509202 | 0.943638 |
| 5 | 0.294118 | 0.464516 | 0.510204 | 0.173913 | 0.019832 | 0.151329 | 0.052519 |
| 6 | 0.176471 | 0.219355 | 0.265306 | 0.271739 | 0.088942 | 0.261759 | 0.072588 |
| 7 | 0.588235 | 0.458065 | 0.489796 | 0.173913 | 0.019832 | 0.349693 | 0.023911 |
| 8 | 0.117647 | 0.987097 | 0.469388 | 0.413043 | 0.635817 | 0.251534 | 0.034159 |
| 9 | 0.470588 | 0.522581 | 0.734694 | 0.173913 | 0.019832 | 0.282209 | 0.065756 |

```
diabetes.describe()
```

| | Num_pregnant | Gluc_concent | Blood_press | Triceps | Insulin | BMI | |
|---|---|---|---|---|---|---|---|
| count | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 768.000000 | 7 |
| mean | 0.226180 | 0.501008 | 0.493742 | 0.221029 | 0.096938 | 0.291430 | |
| std | 0.198210 | 0.196376 | 0.123435 | 0.100315 | 0.126860 | 0.140601 | |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | |
| 25% | 0.058824 | 0.359677 | 0.408163 | 0.173913 | 0.019832 | 0.190184 | |
| 50% | 0.176471 | 0.470968 | 0.489796 | 0.173913 | 0.020733 | 0.282209 | |
| 75% | 0.352941 | 0.620968 | 0.571429 | 0.271739 | 0.136118 | 0.376278 | |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | |

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

```
Ahora los intervalos de las variables se distribuyen entre 0 y 1 tras la normalizació
```

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

```
A las variables Pedrigree y Age se les podría normalizar con una distribución de pears
```

---

## Matriz de correlación

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

```
corr = diabetes.corr()
corr
```

| | Num_pregnant | Gluc_concent | Blood_press | Triceps | Insulin | BMI |
|---|---|---|---|---|---|---|
| **Num_pregnant** | 1.000000 | 0.128213 | 0.208615 | 0.032568 | -0.055697 | 0.021546 |
| **Gluc_concent** | 0.128213 | 1.000000 | 0.218937 | 0.172143 | 0.357573 | 0.231400 |
| **Blood_press** | 0.208615 | 0.218937 | 1.000000 | 0.147809 | -0.028721 | 0.281132 |
| **Triceps** | 0.032568 | 0.172143 | 0.147809 | 1.000000 | 0.238188 | 0.546951 |
| **Insulin** | -0.055697 | 0.357573 | -0.028721 | 0.238188 | 1.000000 | 0.189022 |
| **BMI** | 0.021546 | 0.231400 | 0.281132 | 0.546951 | 0.189022 | 1.000000 |
| **Pedigree** | -0.033523 | 0.137327 | -0.002378 | 0.142977 | 0.178029 | 0.153506 |
| **Age** | 0.544341 | 0.266909 | 0.324915 | 0.054514 | -0.015413 | 0.025744 |
| **Diagnosis** | 0.221898 | 0.492782 | 0.165723 | 0.189065 | 0.148457 | 0.312249 |

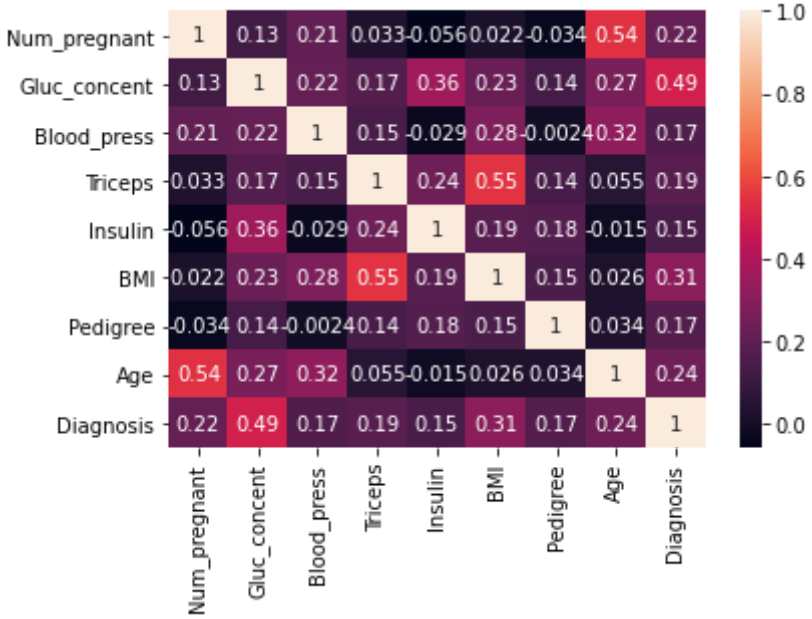
**Haciendo uso de** `seaborn` **podremos hacernos una idea rápida de qué variables están más correlacionadas con la clase. Ejecute el siguiente código.**

```
%matplotlib inline
import seaborn as sns
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureW
  import pandas.util.testing as tm
```

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f64ec7a88d0>
```




### 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.

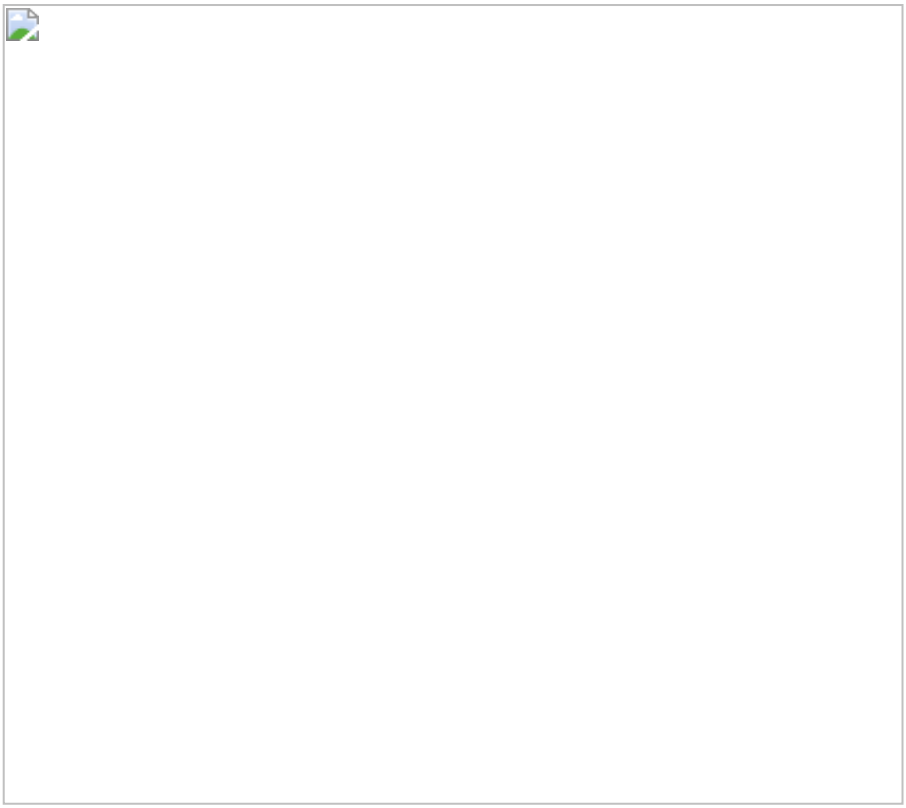
```
La mayor correlación con la clase de diagnosis está en la variable Gluc_content con ur
Sí, tiene sentido pues son las variables que mayor incidencia tienen en la patologìa.
```

## 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:**

```
x_data = diabetes.drop('Diagnosis', axis=1)
```

```
x_data.shape
```

```
(768, 8)
```

```
labels = diabetes['Diagnosis']
```

```
labels.shape
```

```
(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**

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(x_data, labels, test_size=0.33, ra
```

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

```
from sklearn.linear_model import LogisticRegression

diabetesCheck = LogisticRegression(solver='lbfgs')

diabetesCheck.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

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

```
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!**

```
#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',
    median_var = diabetes_original[item].median()
    diabetes_original[item] = diabetes_original[item].replace(to_replace = 0, value =
#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_spli
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, "%")
```

```
  File "<ipython-input-1-bd99fb2a114c>", line 7
    cols_to_reeplace = ['Gluc_concent', 'Blood_press', 'Triceps', 'Insulin',
'BMI', 'Age', 'Pedigree'] for item in cols_to_reeplace:

^
SyntaxError: invalid syntax
```

# 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, "%")**

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

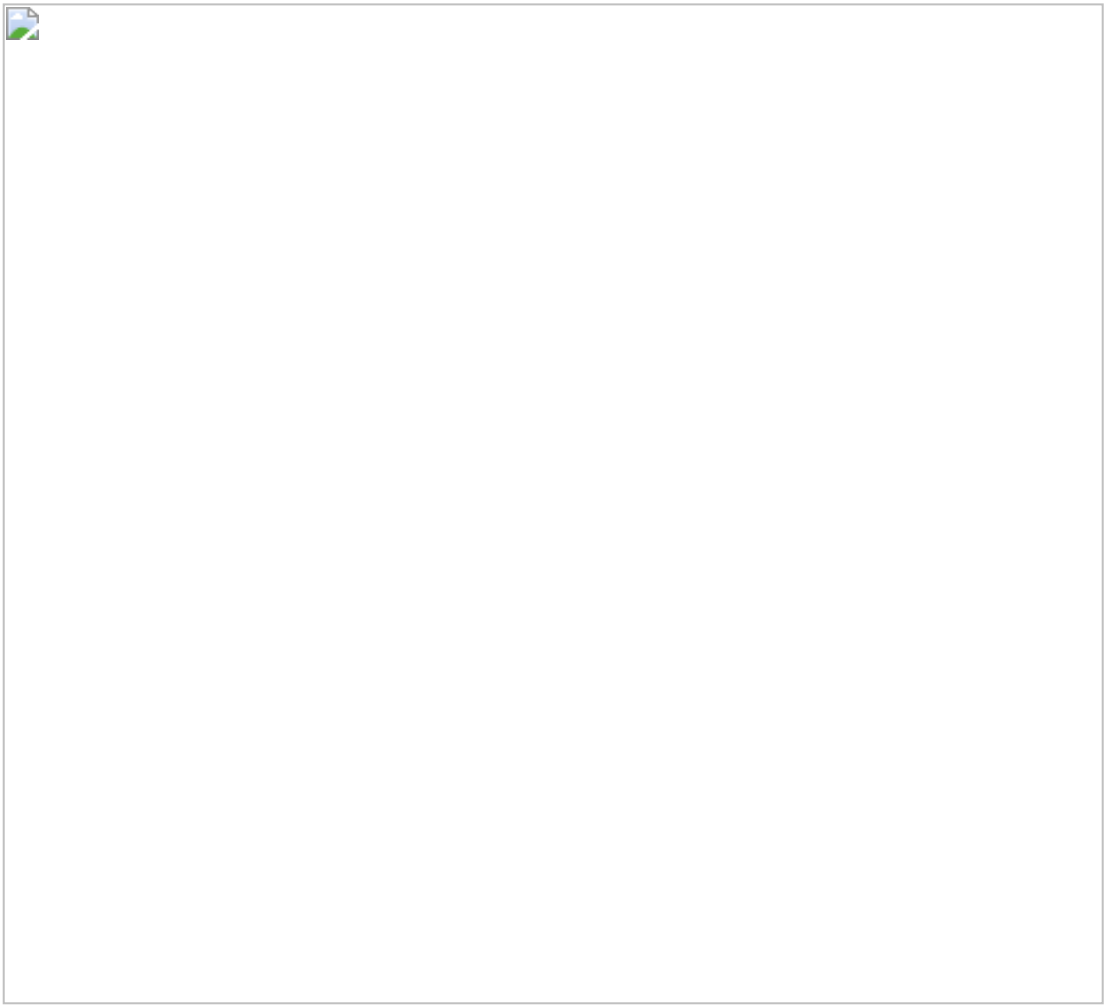
**Tipo de celda no admitido. Haz doble clic para inspeccionar/editar el contenido.**

```
pd.options.display.float_format = '{:,.2f}'.format diabetes_norm = pd.read_csv('data/p
                            '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',
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', '
    diabetes_norm[item]= (diabetes_norm[item] - diabetes_norm[item].mean())/diabetes_n
#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, l
)
#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, "%")
```

```
  File "<ipython-input-2-2f805dcf1dd4>", line 1
    pd.options.display.float_format = '{:,.2f}'.format diabetes_norm =
pd.read_csv('data/pima-indians-diabetes.csv', names=['Num_pregnant',
'Gluc_concent',
                                                                      ^
SyntaxError: invalid syntax
```

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

Se ve mejora escalando o normalizando.
- Con respecto al escalado, debido a que el orden de magnitud de las variables es simi
- Respecto a la normalización, ocurre algo similar al escalado: la mayoría de las cara



**Tipo de celda no admitido. Haz doble clic para inspeccionar/editar el contenido.**