

# Trabajo Práctico RNA - 1C 2019: Prediciendo enfermedad cardiaca en individuos

## EDA (Explanatory Data Analysis)

El objetivo de un EDA es presentar al lector un análisis estadístico de un set de datos. En el presente trabajo, consideramos que hacer uno añade valor al mismo. Explicaremos paso a paso qué se hace en cada tramo de código y brindaremos explicaciones teóricas del mismo.

### 1.- Inspeccionemos el dataset

In [1]:

```
import pandas as pd

df = pd.read_csv('heart.csv')
df.head()
```

Out[1]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	targ
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	

Kaggle nos proporciona la siguiente información (en inglés) de las variables (que en adelante llamaremos `features` ).

- `age` age in years
- `sex` (1 = male; 0 = female)
- `cp` chest pain type
- `trestbps` resting blood pressure (in mm Hg on admission to the hospital)
- `chol` serum cholestoral in mg/dl
- `fbs` (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
- `restecg` resting electrocardiographic results
- `thalach` maximum heart rate achieved
- `exang` exercise induced angina (1 = yes; 0 = no)
- `oldpeak` ST depression induced by exercise relative to rest
- `slope` the slope of the peak exercise ST segment
- `ca` number of major vessels (0-3) colored by flourosopy
- `thal` 3 = normal; 6 = fixed defect; 7 = reversable defect
- `target` 1 or 0

Nuestra variable `y` (dependiente) será `target` ya que como mencionamos, nuestro objetivo es predecir enfermedad cardiaca en individuos en base a las `features` proporcionada por el dataset.

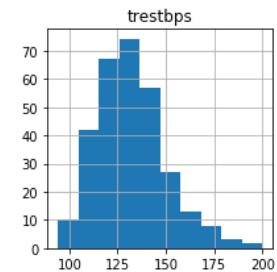
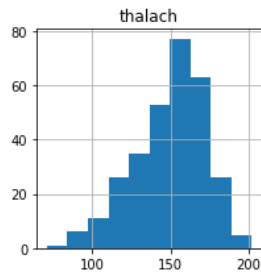
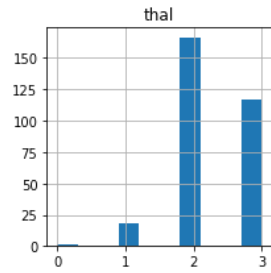
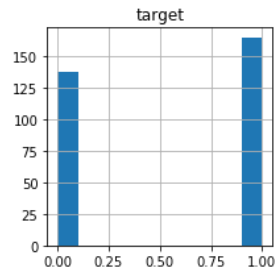
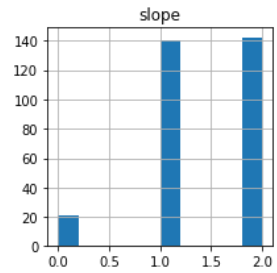
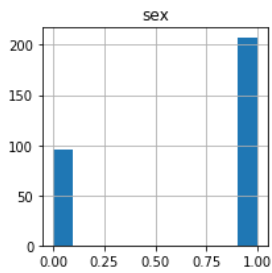
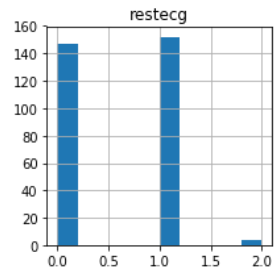
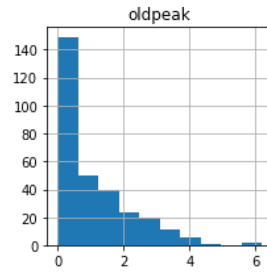
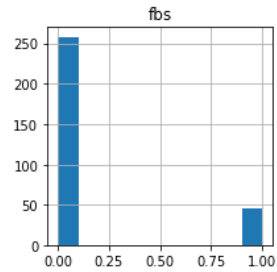
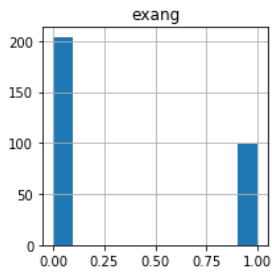
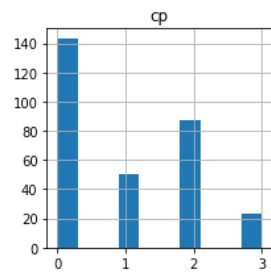
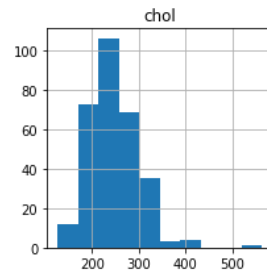
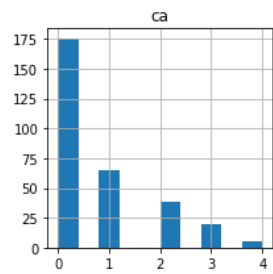
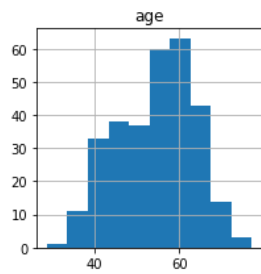
## Veamos la distribución de las variables

In [3]:

```
import matplotlib.pyplot as plt
```

```
df.hist(figsize=(15,15))
```

```
plt.show()
```



Algunos *insights* que vemos de las distribuciones:

- La mayoría de individuos tiene entre 40 y 65 años aproximadamente. Con la media estando en 50-55 años.
- La media de colesterol ronda 250mg/dl. Con una persona por arriba de 500mg/dl 🤖
- Hay mitad de personas que presentan dolor de pecho, la mayoría presenta un dolor tipo 2, que consideramos un dolor "medio".
- La mayoría de los individuos del estudio son hombres.
- La mayoría de los individuos presentan un ritmo cardiaco máximo bastante normal ( *thalach* ), tomando como referencia la imagen debajo (fuente: *American Heart Association*).

Average heart rates by age		
Age in years	Average maximum heart rate in beats per minute	Target heart rate range in beats per minute
40	180	90 to 153
45	175	88 to 149
50	170	85 to 145
55	165	83 to 140
60	160	80 to 136
65	155	78 to 132
70	150	75 to 128
Source: American Heart Association.		

Separamos las features con la variable a predecir.

In [2]:

```
X = df.iloc[:, :-1].values
y = df.iloc[:, -1].values
```

## 2.- Separación del dataset

Dividamos el dataset en un set de entrenamiento y otro para pruebas. Hagamos un 20% para pruebas. Esto nos permite usar el 80% del mismo para entrenar el modelo y el 20% restante para probarlo. Si usáramos el 100% del dataset para entrar y lo probáramos con el mismo dataset, estaríamos cometiendo una suerte de falacia, ya que estamos usando los mismos datos que entraron a la red, para predecir los mismos datos.

In [3]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Procesemos un poco el dataset. Hagamos que la media de cada feature sea 0 y su desviación estándar 1. Esto sirve mucho para algoritmos de machine learning en general, como está descrito en este [artículo](https://en.wikipedia.org/wiki/Feature_scaling) ([https://en.wikipedia.org/wiki/Feature\\_scaling](https://en.wikipedia.org/wiki/Feature_scaling)).

In [4]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
# apply same transformation to test data
X_test = scaler.transform(X_test)
```

## Ahora la parte divertida! 😊

### 3.- Generemos el modelo y alimentémoslo con los datos de entrenamiento

In [5]:

```
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score
```

Probemos diferentes configuraciones de redes neuronales. Vayamos de 1 hidden layer con 3 neuronas a 5 capas con 12 neuronas cada una. Serían 5 x 10 combinaciones.

In [6]:

```
mlp_accuracies_df = pd.DataFrame(columns=['neurons', 'layers', 'score'])

for layers in range(1,5+1):
    for neurons in range(3,12+1):
        clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(neurons, layers), random_state=1)
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        mlp_accuracies_df = mlp_accuracies_df.append({'neurons': neurons, 'layers': layers, 'score': accuracy}, ignore_index=True)
```

In [7]:

```
mlp_accuracies_df.sort_values(by='score', ascending=False)
```

Out[7]:

	neurons	layers	score
27	10.0	3.0	0.901639
28	11.0	3.0	0.885246
9	12.0	1.0	0.885246
15	8.0	2.0	0.868852
14	7.0	2.0	0.868852
12	5.0	2.0	0.868852
46	9.0	5.0	0.852459
39	12.0	4.0	0.836066
30	3.0	4.0	0.836066
22	5.0	3.0	0.836066
49	12.0	5.0	0.836066
43	6.0	5.0	0.836066
6	9.0	1.0	0.836066
47	10.0	5.0	0.836066
38	11.0	4.0	0.819672
40	3.0	5.0	0.819672
44	7.0	5.0	0.819672
36	9.0	4.0	0.819672
18	11.0	2.0	0.819672
32	5.0	4.0	0.819672
20	3.0	3.0	0.819672
21	4.0	3.0	0.819672
24	7.0	3.0	0.819672
34	7.0	4.0	0.803279
48	11.0	5.0	0.803279
37	10.0	4.0	0.803279
25	8.0	3.0	0.803279
11	4.0	2.0	0.803279
2	5.0	1.0	0.803279
23	6.0	3.0	0.803279
1	4.0	1.0	0.803279
17	10.0	2.0	0.803279
13	6.0	2.0	0.803279
16	9.0	2.0	0.786885
3	6.0	1.0	0.786885
5	8.0	1.0	0.786885



	neurons	layers	score
26	9.0	3.0	0.786885
33	6.0	4.0	0.786885
29	12.0	3.0	0.786885
45	8.0	5.0	0.770492
4	7.0	1.0	0.770492
10	3.0	2.0	0.754098
19	12.0	2.0	0.754098
35	8.0	4.0	0.737705
8	11.0	1.0	0.737705
42	5.0	5.0	0.737705
41	4.0	5.0	0.704918
31	4.0	4.0	0.672131
7	10.0	1.0	0.557377
0	3.0	1.0	0.557377

In [11]:

```
clf
```

Out[11]:

```
MLPClassifier(activation='relu', alpha=1e-05, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(6, 5), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='lbfgs', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

## 4.- Analicemos el modelo hecho

Podemos ver los atributos que maneja la clase de Multi-Layer Perceptron Classifier. Posee un `learning_rate` de  $1e-5$  (alpha) y usa como función de activación la ReLU.

En este caso, podemos ver que comparando el `accuracy scoring` (acertados/total) que la configuración de 2 capas escondidas con 5 neuronas cada una (`hidden_layer_sizes`) es la más óptima para este dataset.

A su vez, la documentación de Sklearn nos dice que su clase utiliza *back-propagation* [aquí \(https://scikit-learn.org/stable/modules/neural\\_networks\\_supervised.html#classification\)](https://scikit-learn.org/stable/modules/neural_networks_supervised.html#classification).

## 5.- Veamos qué tan bien va el modelo con los datos de prueba

In [10]:

```
clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(10, 3), random_state=1)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
```

In [11]:

```
cm
```

Out[11]:

```
array([[25,  2],
       [ 4, 30]])
```