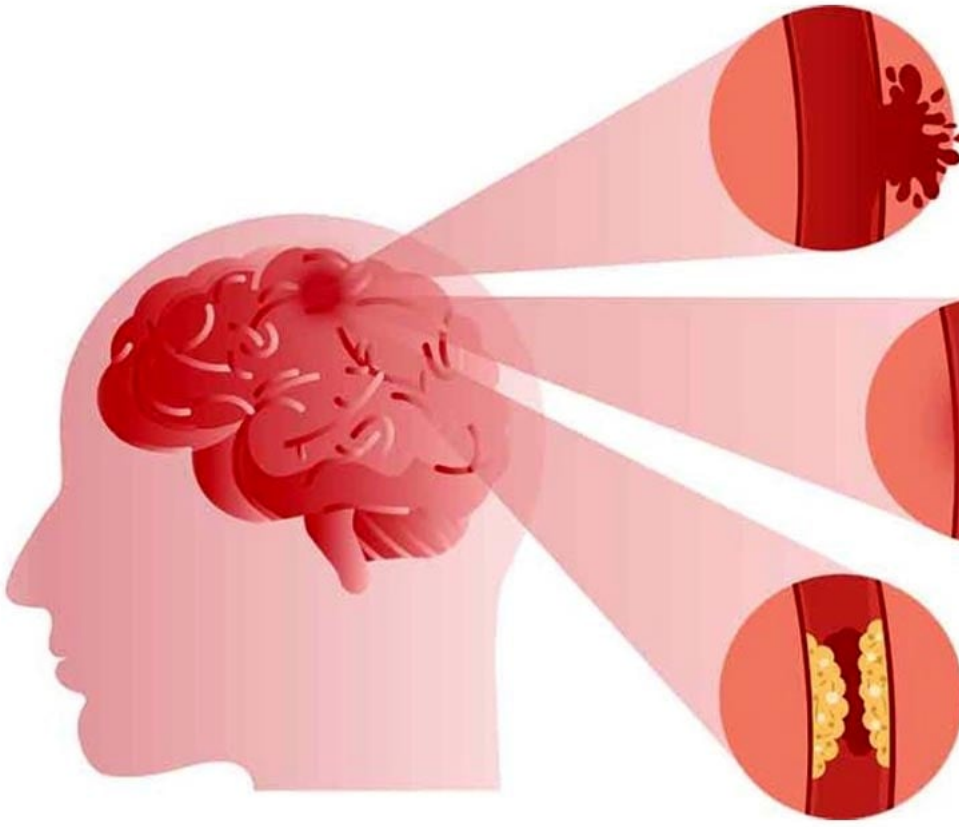
Modulo 2: Implementacion de una tecnica de aprendizaje máquina sin el uso de un framework

Juan Carlos Varela Téllez A01367002
Fecha de inicio: 09/09/2022
Fecha de finalizacion: 09/09/2022

En caso de no tener las bibliotecas necesarias, utilizar los siguientes comandos:
python -m pip install numpy
python -m pip install pandas
python -m pip install seaborn
python -m pip install matplotlib
python -m pip install scikit-learn

Las apoplejias son un evento cuando el suministro de sangre al cerebro se ve interrumpida, causando en falta de oxigeno, daño cerebral y perdida de funciones tanto motoras como mentales.
Globalmente, 1 de cada 4 adultos mayores de 25 años va a tener una apoplejia en su vida.
12,2 millones de personas tendra su primer apoplejia en este año, y 6.5 millones mas moriran como resultado de esta. Mas de 110 millones de personas han tenido una apoplejia.[1]
Este codigo tiene como objetivo analizar datos para poder predecir que personas son mas propensas a tener una apoplejia y asi poder evitar secuelas y bajar estas estadisticas.
[1] https://www.world-stroke.org/world-stroke-day-campaign/why-stroke-matters/learn-about-stroke#:~:text=Globally%201%20in%204%20adults,the%20world%20have%20experienced%20stroke.



Para poder leer, procesar y analizar los datos e información que sacaremos de dichos datos es necesario importar ciertas bibliotecas que nos ayudaran de forma importante:

- Pandas: esta biblioteca nos ayuda a leer nuestros datos, al igual que modificar nuestros datos a traves de un data-frame para manipularlos y analizarlos. Para más información haz click aquí.

- Numpy: esta biblioteca nos da diferentes herramientas matemáticas vectorizadas para acelerar nuestros cálculos. Para más información haz click aquí.
- Scikit-learn: esta biblioteca es de las más importantes que se utiliza ya que contiene la gran mayoría de herramientas de machine learning que se van a utilizar en este reto, desde regresiones hasta bosques aleatorios. Para más información haz click aquí.

Ahora vamos a importar nuestro data-set para poder trabajar. El dat-set se puede encontrar en este link.

Este codigo es una continuacion directa al repositorio https://github.com/JuanVaTe/RetoModulo2, asi que se recomienda revisarlo antes de continuar con este codigo.

Debido a que nuestros modelos de regresion logistica no fueron lo suficientemente complejos para poder dar una prediccion precisa, vamos a utilizar una biblioteca que nos da acceso a herramientas y modelos prehechos que nos ayudaran de forma importante con nuestro problema.

Data-set :chart_with_upwards_trend:

Para poder entender mejor nuestros datos, es necesario saber con que columnas cuenta, asi que para eso vamos a la documentacion del mismo data-set para saber los metadatos.

##Attribute Information

1) gender: "Male", "Female" or "Other"
2) age: age of the patient
3) hypertension: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
4) heart*disease: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease 5) ever*marrie
6) work*type: "children", "Govt*jov", "Never*worked", "Private" or "Self-employed" 7) Residence*type: "Rural" or "Urbar
8) avg*glucose*level: average glucose level in blood
9) bmi: body mass index
10) smoking_status: "formerly smoked", "never smoked", "smokes" or "Unknown"*
11) stroke: 1 if the patient had a stroke or 0 if not

*Note: "Unknown" in smoking_status means that the information is unavailable for this patient

Podemos observar que el data-set tiene 11 caracteristicas, 10 siendo variables independientes y 1 siendo la variable dependiente.

Limpieza de datos y preprocesamiento :factory:

Como se hizo en el codigo pasado, vamos a eliminar las filas que cuentan con valores nulos. En este caso, no hay ningun valor nulo per se, sin embargo, tenemos casillas en la caracteristica de *smoking_status* que tienen el valor `Unknown`, y esto en el contexto de este problema lo podemos contar como un valor nulo. Y ya que estamos limpiando, vamos a cambiar la columna de *Residence_type* a *residence_type'*para tener un formato:

```
stroke_data_clean = stroke_data[stroke_data['smoking_status'] != Unknown'].reset_index(drop=True).rename(columns=
{'Residence_type': residence_type'})
```

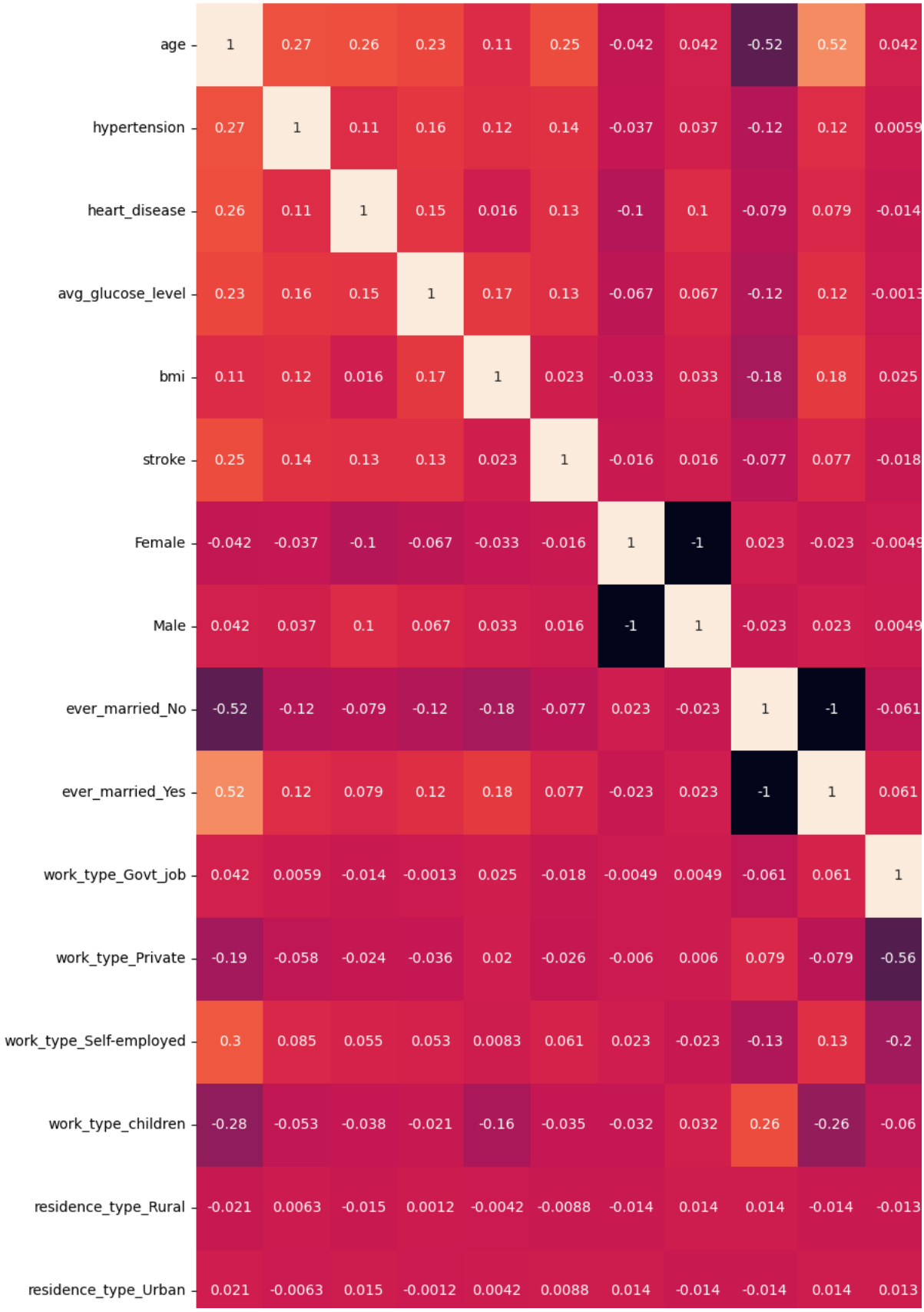
El siguiente paso que tenemos que hacer es un analisis estadistico de este data-set, sin embargo, no se puede hacer de forma correcta debido a las variables cualitativas. Necesitamos cuantificar estas variables. Pandas tiene una funcion que nos va a servir mucho en este caso, la misma funcion que utilizamos en el reto pasado, `get_dummies()`.

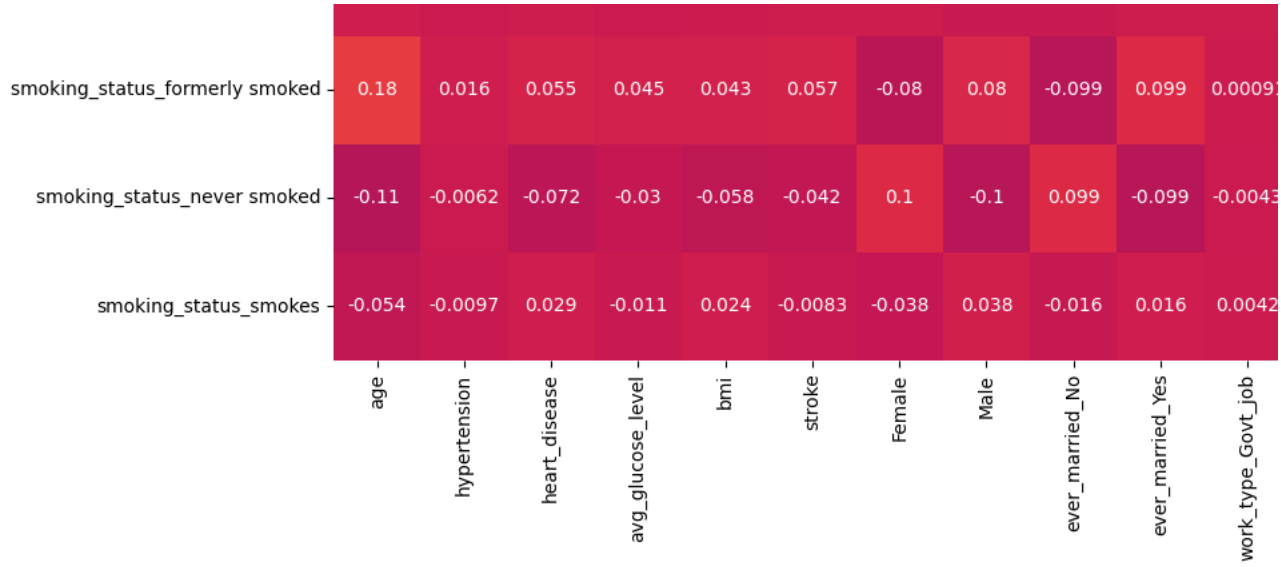
```
# Cuantificamos gender
dummy_gender = pd.get_dummies(stroke_data_clean['gender'])
# Cuantificamos ever_married
dummy_married = pd.get_dummies(stroke_data_clean['ever_married'], prefix='ever_married')
# Cuantificamos work_type
dummy_work_type = pd.get_dummies(stroke_data_clean['work_type'], prefix='work_type')
# Cuantificamos residence_type
dummy_residence_type = pd.get_dummies(stroke_data_clean['residence_type'], prefix='residence_type')
# Cuantificamos smoking_status
dummy_smoking_status = pd.get_dummies(stroke_data_clean['smoking_status'], prefix='smoking_status')

# Se concatena al data-set
stroke_data_clean = pd.concat([stroke_data_clean, dummy_gender, dummy_married, dummy_work_type, dummy_residence_type, dummy

# Por ultimo se eliminan las columnas redundantes
stroke_data_clean = stroke_data_clean.drop(['gender', 'ever_married', 'work_type', 'residence_type', 'smoking_status'], axi
```

Con los datos cuantificados, ahora podemos hacer nuestro analisis estadistico y encontrar correlaciones entre nuestras caracteristicas y nuestra variable dependiente.

```
correlation = stroke_data_clean.corr()
f, ax = plt.subplots(figsize=(20, 20))
sns.heatmap(correlation, annot=True)
plt.show()
```

:arrow_down: :arrow_down: :arrow_down: :arrow_down:


age 1 0.27 0.26 0.23 0.11 0.25 -0.042 0.042 -0.52 0.52 0.042
hypertension 0.27 1 0.11 0.16 0.12 0.14 -0.037 0.037 -0.12 0.12 0.0059
heart_disease 0.26 0.11 1 0.15 0.016 0.13 -0.1 0.1 -0.079 0.079 -0.014
avg_glucose_level 0.23 0.16 0.15 1 0.17 0.13 -0.067 0.067 -0.12 0.12 -0.0013
bmi 0.11 0.12 0.016 0.17 1 0.023 -0.033 0.033 -0.18 0.18 0.025
stroke 0.25 0.14 0.13 0.13 0.023 1 -0.016 0.016 -0.077 0.077 -0.018
Female -0.042 -0.037 -0.1 -0.067 -0.033 -0.016 1 -1 0.023 -0.023 -0.0049
Male 0.042 0.037 0.1 0.067 0.033 0.016 -1 1 -0.023 0.023 0.0049
ever_married_No -0.52 -0.12 -0.079 -0.12 -0.18 -0.077 0.023 -0.023 1 -1 -0.061
ever_married_Yes 0.52 0.12 0.079 0.12 0.18 0.077 -0.023 0.023 -1 1 0.061
work_type_Govt_job 0.042 0.0059 -0.014 -0.0013 0.025 -0.018 -0.0049 0.0049 -0.061 0.061 1
work_type_Private -0.19 -0.058 -0.024 -0.036 0.02 -0.026 -0.006 0.006 0.079 -0.079 -0.56
work_type_Self-employed 0.3 0.085 0.055 0.053 0.0083 0.061 0.023 -0.023 -0.13 0.13 -0.2
work_type_children -0.28 -0.053 -0.038 -0.021 -0.16 -0.035 -0.032 0.032 0.26 -0.26 -0.06
residence_type_Rural -0.021 0.0063 -0.015 0.0012 -0.0042 -0.0088 -0.014 0.014 0.014 -0.014 -0.013
residence_type_Urban 0.021 -0.0063 0.015 -0.0012 0.0042 0.0088 0.014 -0.014 -0.014 0.014 0.013



Gracias a la grafica podemos apreciar que el factor mas importante entre todos, al menos en un aspecto estadistico e individual, es la edad, seguido por la hipertension y enfermedades del corazon y el nivel de glucosa.
Para el preprocesamiento de datos, vamos a quedarnos con 2 data-sets de variables independientes:

- Data-set con solamente las caracteristicas mas correlacionadas con la apopplejia (`age`, `hypertension`, `heart_disease`, `avg_glucose_level`)
- Data-set con todas las caracteristicas
  Esto es para tener mas espacio de experimentacion al momento de utilizar los modelos de machine learning.

Por ultimo, vamos a escalar todos los datos. Esto es para ayudar a que nuestros modelos encuentren de una forma mas facil la convergencia de la funcion de costo.

```
# Escalador para data-frame completo
escalador_all = StandardScaler()
escalador_all.fit(data_x)
data_x_scaled = pd.DataFrame(escalador_all.transform(data_x))

# Escalador para data-frame con variables correlacionadas
escalador_correlation = StandardScaler()
escalador_correlation.fit(data_x_correlation)
data_x_correlation_scaled = pd.DataFrame(escalador_correlation.transform(data_x_correlation))
```

Ahora vamos a modularizar los datos.
Por buena practica es necesario tener 3 modulos: entrenemiento, validacion y pruebas.
Por esta ocasion, el modulo de pruebas tendra 5 filas, y sera un subconjunto del modulo de validacion.

Prueba de modelos de Machine Learning :computer:

El metodo para poder conseguir la mejor combinacion de hiperparametros es un proceso iterativo ya que se necesita de mucha experimentacion para poder afinar de forma correcta a los modelos.
Se empezara con un modelo sin hiperparamtros, con los valores default que tenga la libreria de *scikit-learn*. Esto normalmente genera un modelo muy complejo con overfitting. A partir de el resultado que nos de, se empezara a afinar con diferentes hiperparametros, dependiendo de si queremos aumentar o disminuir la complejidad del modelo.

Regresion logistica :chart_with_downwards_trend:

Empecemos con un modelo sencillo, que es el de regresion logistica.

```
logistic_regression = LogisticRegression(random_state=0)
logistic_regression.fit(train_x_all, train_y_all)
```

```
Regresion logistica ================================================
Puntaje de entrenamiento: 0.9413793103448276
Puntaje de validacion: 0.9460390355912744
================================================
```

Curiosamente, desde la primera iteracion de experimentacion del modelo dio un resultado muy alto. Se intento con ambos data-sets, cambiando el optimizador, diferentes hiperparametros del modelo, pero al final, el default fue el que mayor puntaje tuvo.

Arbol de decision :deciduous_tree:

Ahora utilizaremos un arbol de decision.
Los árboles de decisión llegan a ser muy propensos al overfitting, es por esto que tenemos que experimentar mucho con los hiperparámetros.

Tambien no se cuenta con una fórmula mágica para escoger los mejores hiperparametros, es por eso que la experimentación y llegar a un equilibrio es muy importante.

```
decision_tree = DecisionTreeClassifier(random_state=0,
                                                                    max_depth=12)
decision_tree.fit(train_x_corr, train_y_corr)
```

```
Arbol de decision sin podar ======================================
Puntaje de entrenamiento: 0.9731800766283525
Puntaje de validacion: 0.9322617680826636
Profundidad maxima: 12
================================================
```

```
pruning_data = decision_tree.cost_complexity_pruning_path(train_x_corr, train_y_corr)

best_alpha = 0
best_score = 0

# Probando todas las alfas
for alpha in pruning_data.ccp_alphas:
        tree = DecisionTreeClassifier(random_state=0,
                                                                max_depth=12,
                                                                ccp_alpha=alpha)
        tree.fit(train_x_corr, train_y_corr)

        if tree.score(test_x_corr, test_y_corr) > best_score:
                best_score = tree.score(test_x_corr, test_y_corr)
                best_alpha = alpha

# Nuevo arbol con la mejor alfa
decision_tree = DecisionTreeClassifier(random_state=0,
                                                                    max_depth=12,
                                                                    ccp_alpha=best_alpha)
decision_tree.fit(train_x_corr, train_y_corr)
```

```
Arbol de decision podado ======================================
Puntaje de entrenamiento: 0.9409961685823754
Puntaje de validacion: 0.9460390355912744
Mejor alfa: 0.00088584501076195
=======================================================
```

Nuestro valor alfa básicamente nos dice que partes del árbol "podar" debido a que indica el valor donde nuestros puntajes cambian, esto nos ayuda a generalizar mejor.
Debido a que se nos devolvió una gran cantidad de alfas, tuvimos que probarlas todas.
En este ejemplo, lo mejor fue utilizar el data-set con solamente los datos correlacionados. Esto ayuda a generalizar mejor, ya que los arboles de decision tienden a tener overfitting.

Algo que me parece muy raro es el hecho de que se obtiene el mismo puntaje de validacion que la regresion logistica, es casi como si los mismos datos nos dijeran que esa es la mejor puntacion que podemos obtener.

Bosque aleatorio :palm_tree: :evergreen_tree: :deciduous_tree:

Ahora utilizaremos un modelo de bosque aleatorio, que basicamente es el promedio de resultados de un conjunto de arboles de decision, de ahi el nombre de bosque.

```
random_forest = RandomForestClassifier(random_state=0,
                                                                    n_estimators=1000,
                                                                    max_depth=11,
                                                                    min_samples_leaf=5)
random_forest.fit(train_x_all, train_y_all)
```

```
Bosque aleatorio ================================================
Puntaje de entrenamiento: 0.9409961685823754
Puntaje de validacion: 0.9460390355912744
================================================
```

En este caso fue mejor utilizar el data-set con todos los datos. El bosque aleatorio tiene medidas para contrarrestar el overfitting, es por esto que utilizar este data-set obtiene mejores resultados.
Una vez mas aparecio este puntaje, el mismo que los otros modelos anteriores, sacados con data-sets diferentes. Es bastante curioso esto.
Por ultimo quisiera remarcar que este bosque aleatorio no fue el que tuvo mejor puntaje en su entrenamiento, sin embargo, se prefiere que el puntaje de la validacion sea mas alto ya que esto indica que el modelo generaliza de una mejor manera, tanto asi que predice mejor con datos nuevos que con datos que utilizo para entrenar.

Red neuronal :globe_with_meridians:

Es turno de uno de los modelos más robustos y consistentes de machine learning, las redes neuronales.
Este modelo cuenta con una gran cantidad de hiperparámetros asi que escoger de forma informada los hiperparametros para afinar este modelo es importante.

Red neuronal con data-set completo

```
neural_network_full = MLPClassifier(random_state=0,
                                                                    hidden_layer_sizes=(8, 8),
                                                                    learning_rate='adaptive',
                                                                    max_iter=10000)
neural_network_full.fit(train_x_all, train_y_all)
```

```
Red neuronal con data-set completo ============================
Puntaje de entrenamiento: 0.9421455938697318
Puntaje de validacion: 0.9460390355912744
================================================
```

Red neuronal con data-set correlacional

```
neural_network_corr = MLPClassifier(random_state=0,
                                                                    hidden_layer_sizes=(8, 8),
                                                                    learning_rate='adaptive',
                                                                    max_iter=10000)
neural_network_corr.fit(train_x_corr, train_y_corr)
```

```
Red neuronal con data-set correlacional ========================
Puntaje de entrenamiento: 0.9409961685823754
Puntaje de validacion: 0.9460390355912744
================================================
```

Por la naturaleza de una red neuronal, utilizar uno u otro data-set no tiene tanta diferencia como los otros modelos. Esto es porque este cambio de pesos para cada caracteristica es algo intrinseco en la red neuronal, mejorando con cada iteracion al darle mas importancia a las caracteristicas que la necesitan.
No hay mejor demostracion de esto que el hecho de que ambas rede neuronales hayan llegado al mismo puntaje de validacion y un puntaje extremadamente similar en su puntaje de entrenamiento, aunque en algunos casos tambien quedo igual.
Es por esta razon por la cual quise dejar ambas redes neuronales, para demostracion de como se comporta una red neuronal.

Comparacion de rendimientos :bar_chart:

Por ultimo, vamos a hacer unas comparaciones de modelos con matrices de confusion ya que ademas de dar metricas de rendimiento, tambien nos sirve para ver cuantas predicciones fueron correctas y cuantas no.

```
# Definicion de funcion para sacar las metricas de rendimiento
def metricas_rendimiento(matriz_confusion):
        exactitud = (matriz_confusion[0][0] + matriz_confusion[1][1]) / (
                                matriz_confusion[0][0] + matriz_confusion[0][1] + matriz_confusion[1][0] + matriz_confusion

        try:
                precision = matriz_confusion[0][0] / (matriz_confusion[0][0] + matriz_confusion[1][0])
        except:
                precision = 0

        exhaustividad = matriz_confusion[0][0] / (matriz_confusion[0][0] + matriz_confusion[0][1])

        try:
                puntaje_F1 = (2 * precision * exhaustividad) / (precision + exhaustividad)
        except:
                puntaje_F1 = 0

        return exactitud, precision, exhaustividad, puntaje_F1

# Listado de modelos para ciclar la obtencion de metricas y matrices de confusion

models = [['Regresion Logistica', 'all', logistic_regression],
                  ['Arbol de decision', 'corr', decision_tree],
                  ['Bosque aleatorio', 'all', random_forest],
                  ['Red neuronal con data-set completo', 'all', neural_network_full],
                  ['Red neuronal con data-set correlacioal', 'corr', neural_network_corr]]

for trio in models:
        if trio[1] == 'all':
                conf_matrix = confusion_matrix(test_y_all, trio[2].predict(test_x_all))
        else:
                conf_matrix = confusion_matrix(test_y_corr, trio[2].predict(test_x_corr))

        acc, prec, recall, F1_score = metricas_rendimiento(conf_matrix)

        print("================================================")
        print(f"Metricas de rendimiento para modelo de {trio[0]}")
        print("Matriz de confusion:")
        print(conf_matrix)
        print(f"Exactitud     : {acc}")
```

```
        print(f"Precision     : {prec}")
        print(f"Exhaustividad : {recall}")
        print(f"Puntaje F1    : {F1_score}")
        print("==========================================\n")
```

```
==========================================
Metricas de rendimiento para modelo de Regresion Logistica
Matriz de confusion:
[[824   0]
 [ 47   0]]
Exactitud     : 0.9460390355912744
Precision     : 0.9460390355912744
Exhaustividad : 1.0
Puntaje F1    : 0.9722713864306785
==========================================


==========================================
Metricas de rendimiento para modelo de Arbol de decision
Matriz de confusion:
[[824   0]
 [ 47   0]]
Exactitud     : 0.9460390355912744
Precision     : 0.9460390355912744
Exhaustividad : 1.0
Puntaje F1    : 0.9722713864306785
==========================================
```

```
==========================================
Metricas de rendimiento para modelo de Bosque aleatorio
Matriz de confusion:
[[824   0]
 [ 47   0]]
Exactitud     : 0.9460390355912744
Precision     : 0.9460390355912744
Exhaustividad : 1.0
Puntaje F1    : 0.9722713864306785
==========================================


==========================================
Metricas de rendimiento para modelo de Red neuronal con data-set completo
Matriz de confusion:
[[824   0]
 [ 47   0]]
Exactitud     : 0.9460390355912744
Precision     : 0.9460390355912744
Exhaustividad : 1.0
Puntaje F1    : 0.9722713864306785
==========================================
```

```
============================================
Metricas de rendimiento para modelo de Red neuronal con data-set correlacioal
Matriz de confusion:
[[824   0]
 [ 47   0]]
Exactitud     : 0.9460390355912744
Precision     : 0.9460390355912744
Exhaustividad : 1.0
Puntaje F1    : 0.9722713864306785
============================================
```

Aqui es donde se acaba la magia.
La razon por la cual nuestros modelos se parecian tanto es porque todos nuestros modelos se inclinan demasiado a decir que no hay peligro de apoplejia, tanto asi que para estos modelos no es posible que a alguien le de una apoplejia.
Sin embargo, los modelos utilizados aun nos dejan con el aprendizaje de una biblioteca muy completa. En un futuro cercano, se intentara utilizar nuevamente esta biblioteca para poder hacer predicciones mucho mas certeras.

Predicciones :stars:

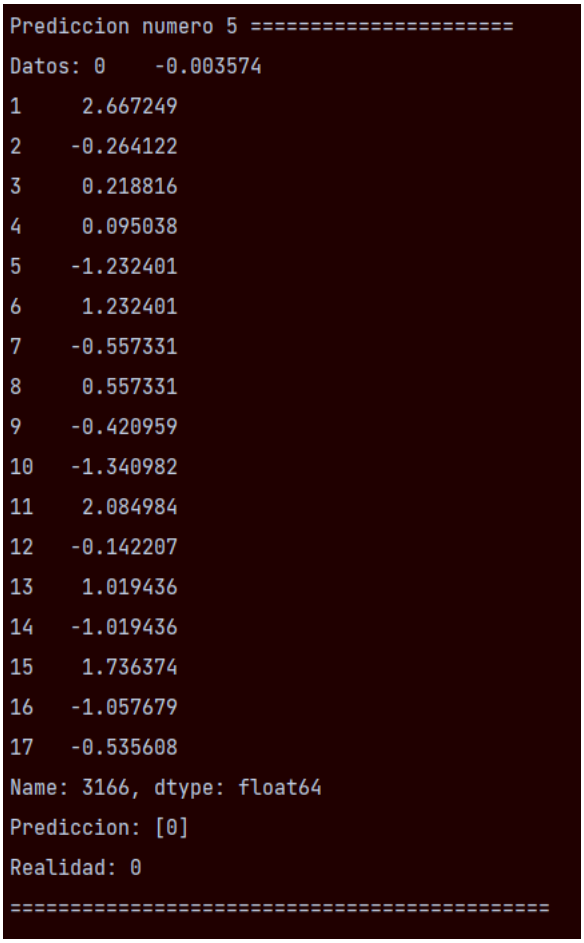
Por ultimo, vamos a hacer unas predicciones con la red neuronal que utilizo el data-set completo para poder validar su rendimiento:

```
Prediccion numero 1 ======================
Datos: 0     1.531505
1     2.667249
2    -0.264122
3     1.976057
4     2.551916
5     0.811424
6    -0.811424
7     1.794267
8    -1.794267
9    -0.420959
10    0.745722
11   -0.479620
12   -0.142207
13   -0.980934
14    0.980934
15   -0.575913
16    0.945466
17   -0.535608
Name: 117, dtype: float64
Prediccion: [0]
Realidad: 1
==========================================
```

```
Prediccion numero 2 ======================
Datos: 0    -1.591586
1    -0.374918
2    -0.264122
3    -0.704583
4    -0.226122
5    -1.232401
6     1.232401
7     1.794267
8    -1.794267
9    -0.420959
10    0.745722
11   -0.479620
12   -0.142207
13   -0.980934
14    0.980934
15   -0.575913
16    0.945466
17   -0.535608
Name: 1140, dtype: float64
Prediccion: [0]
Realidad: 0
==========================================
```

```
Prediccion numero 3 ======================
Datos: 0    -0.797580
1    -0.374918
2    -0.264122
3    -0.868733
4    -0.595457
5    -1.232401
6     1.232401
7    -0.557331
8     0.557331
9    -0.420959
10    0.745722
11   -0.479620
12   -0.142207
13   -0.980934
14    0.980934
15   -0.575913
16    0.945466
17   -0.535608
Name: 2426, dtype: float64
Prediccion: [0]
Realidad: 0
==========================================
```

```
Prediccion numero 4 ======================
Datos: 0    -0.903447
1    -0.374918
2    -0.264122
3    -0.158731
4     2.262872
5     0.811424
6    -0.811424
7    -0.557331
8     0.557331
9     2.375527
10   -1.340982
11   -0.479620
12   -0.142207
13   -0.980934
14    0.980934
15    1.736374
16   -1.057679
17   -0.535608
Name: 2564, dtype: float64
Prediccion: [0]
Realidad: 0
==========================================
```



Esto nos ensena que en efecto, los modelos estan tan inclinados a decir que es imposible tener una apoplejia, no importa tu perfil medico.

Conclusion :white_check_mark:

Para poder crear un modelo de Machine Learning, no solamente puedes agarrar un monton de datos y esperar a que el modelo sea el mejor posible.
Hay que hacer un proceso para poder tener un modelo que en verdad cumpla nuestras expectativas.
Para poder obtener una vista mas detallada del proceso detras de escenas, puedes revisar el codigo aqui.

Mejoras a partir de la retroalimentacion

- Creacion de documentacion en README.md
- Creacion de documentacion en .pdf
- Implementacion de predicciones explicitas