ESTUDIANTES:

- JONATTAN STIVENT VARGAS CAMACHO - 2142612
- JONATHAN BUITRAGO - 2163024

PROYECCIÓN DE COSTOS DE LA ATENCIÓN HOSPITALARIA EN BUCARAMANGA POR ACCIDENTES DE TRÁNSITO

INTRODUCCION:

EVALUAREMOS DATOS QUE CONTIENEN COSTOS PROMEDIOS DE ACCIDENTES DE TRANSITO DE ENERO 2018 A OCTUBRE 2019. SE ENCUENTRAN DATOS COMO TIPO DE VEHÍCULO, RELACIÓN AL CONDUCTOR, SEXO, FECHA DEL SINIESTRO, EPS Y DEMÁS DATOS IMPORTANTES. DATOS PROVISTOS POR https://www.datos.gov.co/, DATOS ABIERTOS, GOBIERNO DE COLOMBIA.

DATASET: https://www.datos.gov.co/Salud-y-Protecci-n-Social/Costos-de-la-atenci-n-hospitalaria-en-Bucaramanga-/g4vd-w4ip

1. MOTIVACIÓN PARA EL DESARROLLO DEL PROYECTO:

PRESTAR UN SERVICIO PARA PREDECIR CON LA MAYOR EXACTITUD POSIBLE EL COSTO DE LA ATENCIÓN HOSPITALARIA DE ACCIDENTES DE TRÁNSITO EN B/GA

2. TEMA PRINCIPAL DE INTELIGENCIA ARTIFICIAL ABORDADO

MACHINE LEARNING: REGRESIÓN

SOPORTADO POR: scikit learn NumPy

3. FUNCIONAMIENTO Y SIMULACIÓN DEL PROYECTO

Para la representación y simulación del proyecto nos basamos en el notebook, en este caso se utilizó la herramienta **Colaboratory** proporcionada por Google en donde se está gestionando el proyecto general. Además en el notebook se encontrará una breve explicación de las acciones pertinentes para realizar adecuadamente cada uno de los objetivos del proyecto.

1. **INSTANCIACIÓN INICIAL**
   Para iniciar el proyecto, como se está acostumbrado, importamos cada uno de los módulos que se usarán para el funcionamiento del mismo.

```
from __future__ import division, print_function, unicode_literals #py2 and py3
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import *

from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC

from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR

from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline

import tensorflow as tf
from tensorflow import keras
import os
import sys

from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN
```

2. UTILIZACIÓN DE RECURSOS

Se realiza la llamada a cada uno de los recursos que se usarán, en este caso el dataset, donde se hace una muestra de cinco registros.

```
d = pd.read_csv("../data/costos_atencion_hospitalaria.csv")

pd.set_option('display.width', 500)
pd.set_option('precision', 5)
pd.set_option('display.max_columns',25)

print("Dimensión de los datos:", d.shape)
print(d.head(9))
```

Resultado:

```
Dimensión de los datos: (19643, 24)
   Numero  EDAD GRUPO ETAREO  CURSO DE VIDA       SEXO TIPO DE VEHÍCULO RELACION USUARIO/ACCIDENTE   FECHA DE ACCIDENTE HORA DE ACCIDENTE DIA ACCI
0       1    68     60 y mas  Persona Mayor  Masculino             Moto                  Conductor  07/01/2018 18:00:00            18: 00
1       2    25      19 a 28        Jovenes   Femenino             Moto                  Conductor  01/01/2018 00:50:00            00: 50
2       3    20      19 a 28        Jovenes  Masculino             Moto                  Conductor  03/01/2018 07:00:00            07: 00
3       4    17      12 a 18   Adolescencia  Masculino             Moto                  Conductor  07/01/2018 18:00:00            18: 00
4       5    29      29 a 59        Adultez   Femenino             Moto                  Conductor  12/01/2018 07:00:00            07: 00
5       6    34      29 a 59        Adultez   Femenino             Moto                  Conductor  13/01/2018 07:40:00            07: 40
6       7    22      19 a 28        Jovenes  Masculino            Carro                  Conductor  14/01/2018 01:30:00            01: 30
7       8    23      19 a 28        Jovenes  Masculino             Moto                  Conductor  03/01/2018 04:30:00            04: 30
8       9    31      29 a 59        Adultez  Masculino              Bus                     Peatón  13/01/2018 17:30:00            17: 30
```

3. PROCESAMIENTO DE Y PREPARACIÓN DE LA INFORMACIÓN

Se eliminan las columnas que no servirán ya que hacer uso de ellas no es necesariamente adecuado, pertinente ni tiene propósito alguno para el sistema de información por parte de un usuario que tiene como objetivo final predecir el costo que tendría su supuesto accidente, como por ejemplo FECHA_DE_ACCIDENTE.

Recordar que la información en todo sistema de información debe tener las propiedades de Pertinencia y Propósito.

```
del d["Numero"], d['GRUPO ETAREO'], d['CURSO DE VIDA'], d['FECHA DE ACCIDENTE'], d['HORA DE ACCIDENTE'], d['DIA ACCIDENTE'], d['AÑO ACCIDENTE']
del d['FECHA DE INGRESO IPS'], d['HORA DE INGRESO IPS'], d['DIA DE INGRESO IPS'], d['ANO INGRESO IPS']

d.rename(columns={'TIPO DE VEHÍCULO':'TIPO_DE_VEHÍCULO','RELACION USUARIO/ACCIDENTE':'RELACION_USUARIO_ACCIDENTE',
                  'DIA SEMANA ACCIDENTE':'DIA_SEMANA_ACCIDENTE','MES ACCIDENTE':'MES_DEL_ACCIDENTE','REQUERIMIENTO DE CX':'CIRUGIA',
                  'REFERIDO A OTRA IPS':'REFERIDO_A_OTRA_IPS','CONDICION EGRESO':'CONDICION_EGRESO','COSTOS':'COSTO_MÉDICO',
                  'EPS USUARIO':'EPS','DIA SEMANA DE INGRESO IPS':'DIA_SEMANA_INGRESO_IPS',
                  'MES DE INGRESO IPS':'MES_INGRESO_IPS'},inplace=True)

print( "Dimensión de los datos:", d.shape)
print(d.head(3))
```

Aplicando este mismo principio y además realizando la eliminación de otro tipo de información corrupta se llega al objetivo inicial que es la preparación de un dataset con información útil y sin datos indeterminados ya que esto causaría problemas en la puntuación y en el modelo que se usaría para entrenar.

4. PROCESO DE REGRESIÓN

El valor principal de la predicción que se desea generar es el valor del costo a partir de cada una de las características, por ende se usará un regresor ya que es el método más aproximado para obtener o generar información a partir de una serie de características.

```
[ ]  1 X = d.values[:,[0,1,2,3,4,5,6,7,8,10,11,12]]
     2 y2 = d.values[:,[9]]
     3 y = y2.ravel()
     4 test_size = 0.5
     5 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)
     6 estimador = RandomForestRegressor().fit(X_train, y_train)
     7 estimador.score(X_train, y_train)
```

Por ende tenemos dos variables que se usarán para obtener la información del dataset. La variable **X** hace referencia a todas las doce características resultantes después de una exhaustiva eliminación y verificación de información. Por otro lado la variable **y** hará referencia al **Ground Truth** que en este caso es el costo.
Además se usó el método **train_test_split()** que es el encargado de la separación de los datos dado un porcentaje del mismo, en este caso se realizó la separación del 50%.
Por último se hace la instanciación del método **RandomForestRegressor(),** en donde se entrena tanto con las características como con el ground truth, en donde se tiene una puntuación final de `0.8792261696931112`, es decir, **87,92%.** Esta puntuación final es a partir de pasar los datos de entrenamiento al estimador de la puntuación, por esa razón es una buena puntuación. ¿Qué pasaría si se usaran los datos de testeo?. Se obtiene una puntuación de `0.15088062189551177`, es decir, **15,08%,** una muy mala predicción.

5. USO DE PCA

Para obtener un dato más satisfactorio (ya que ocurre en ocasiones) use usó un PCA para reducir características debido a que se contaba con 12 características y el objetivo era llegar a la cantidad de características donde se obtuviera la mejor puntuación, en la que se llegó a un total de 10 características.

```
[ ]  1 X = d.values[:,[0,1,2,3,4,5,6,7,8,10,11,12]]
     2 X = PCA(n_components=10).fit_transform(X)
     3 y2 = d.values[:,[9]]
     4 y = y2.ravel()
     5 test_size = 0.2
     6 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)
     7 estimador = RandomForestRegressor().fit(X_train, y_train)
     8 estimador.score(X_test, y_test)
```

En donde pasando como parámetros al estimador y su puntuación los datos de testeo, se obtuvo una puntuación final de `0.20010252352376934`, es decir **20.01%.** Un resultado de igual manera no deseado pero que sí tuvo una mejora respecto al modelo sin PCA. Sin embargo también se realizó el análogo con otro número de características, cabe recalcar que en este caso se usó **80%** de la información para entrenamiento y el **20%** de la información se usó para testeo. Usando un PCA con 11 características de las 12 iniciales se obtuvo una puntuación de `-0.061551037451338386`, es decir -0.6%, por lo que es una puntuación pésima.

6. COMPARACIÓN CON OTROS MÉTODOS

Uso de una máquina de soporte vectorial.

```
[ ]  1 X = d.values[:,[0,1,2,3,4,5,6,7,8,10,11,12]]
     2 X = PCA(n_components=10).fit_transform(X)
     3 y2 = d.values[:,[9]]
     4 y = y2.ravel()
     5 test_size = 0.2
     6 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)
     7 estimador = SVR(kernel="linear").fit(X_train, y_train)
     8 estimador.score(X_test, y_test)
```

Uso de un modelo lineal.

```
[ ]  1 from sklearn import linear_model
     2
     3 X = d.values[:,[0,1,2,3,4,5,6,7,8,10,11,12]]
     4 X = PCA(n_components=10).fit_transform(X)
     5 y2 = d.values[:,[9]]
     6 y = y2.ravel()
     7 test_size = 0.2
     8 X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)
     9 estimador = linear_model.LassoLars(alpha=.8).fit(X_train, y_train)
    10 estimador.score(X_test, y_test)
```

Como resultado se obtiene este último una mejoría dando un resultado `0.23038338635061362`, es decir, **23,03%.**

A partir de este último modelo se realizó una investigación del mismo, en donde se habla en la documentación de la existencia de parámetros para la configuración de este. De esta investigación se obtuvo un parámetro llamado 'alpha' (Fuerza de regularización; debe ser un flotador positivo. La regularización mejora el condicionamiento del problema y reduce la varianza de las estimaciones).

```
from sklearn import linear_model

X = d.values[:,[0,1,2,3,4,5,6,7,8,10,11,12]]
X = PCA(n_components=10).fit_transform(X)
y2 = d.values[:,[9]]
y = y2.ravel()
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)
estimador = linear_model.Lasso(alpha=0.7).fit(X_train, y_train)
estimador.score(X_test, y_test)
```

Teniendo un Alpha=0.7 se obtiene una puntuación de `0.265518571869060 25`, es decir, **26.55%,** se nota una mejora respecto a los anteriores. No obstante se siguió variando este valor entre 0 y 1 siendo 0.7 el mejor valor del parámetro.

7. **CONCLUSIÓN REGRESOR**

A partir de la puntuación tan baja obtenida, se analizó otra situación para llegar al problema de la regresión del valor de precios, a partir de lo anterior se realizó la regresión para otro ground truth, obteniendo así puntuaciones más considerables y de mayor utilidad.

```
X = d.values[:,:-1]
X = PCA(n_components=11).fit_transform(X)
y2 = d.values[:,-1]
y = y2.ravel()
test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)
estimador = RandomForestRegressor().fit(X_train, y_train)
estimador.score(X_test, y_test)
```

En este caso haciendo uso de PCA y de otro ground truth pero con la misma cantidad de características se obtuvo una puntuación de `0.9558692322905954`, es decir, **95,58%,** un incremento en el valor más alto por lo que se llegó a una conclusión y es la cantidad de dígitos del valor. Esto quiere decir que como las cantidades de precios son altas y con grandes cantidades de dígitos, el desface que existe entre estas generan una mala puntuación o en otras palabras, si tenemos un valor de 5,000,000 pero el regresor dio un valor de 5,012,123 estos últimos dígitos dañan la puntuación del regresor, por lo que se decidió obviar toda la cantidad de dígitos y sólo se tomó una parte más importante o los dígitos con mayor peso.

8. **USO DE REDES NEURONALES**

Para tener una mayor precisión o buscar otra solución, también se utilizaron redes neuronales para verificar la puntuación o si tal vez tenía algo mejor que ofrecernos que la regresión.

```
X = d.values[:,[0,1,2,3,4,5,6,7,8,10,11,12]]
y2 = d.values[:,[9]]
y = y2.ravel()

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=test_size)

XX = np.r_[X]
yy = np.r_[y]
print('XX:', XX.shape)
print('yy:', yy.shape)
```

Se pretende entrenar con las mismas características y el ground truth inicial (costo).

```
[ ]  model = keras.models.Sequential([
         tf.keras.layers.InputLayer(input_shape=(12,)),
         keras.layers.Dense(128),
         keras.layers.Dense(14405231, activation=tf.nn.softmax)
     ])

[ ]  model.compile(optimizer=tf.train.AdamOptimizer(),
                   loss='sparse_categorical_crossentropy',
                   metrics=['accuracy'])

[ ]  model.fit(X_train, y_train, batch_size=10, epochs=15)

     Train on 9029 samples
     Epoch 1/15
```

De forma obligatoria se debía dejar ingresado el numero 14405231 en la capa Dense puesta para generar el funcionamiento pero Google Collaboratory no tiene el soporte para la red neural puesta (aviso de la plataforma Google Collaboratory), así que nos quedamos con los regresores anteriores.

9. **USO KMEANS Y DBSCAN**

Además se trató de implementar dos algoritmos no supervisados, esto con el fin de realizar la comparación entre precios y EPS, es decir, a partir de los clústeres cuáles son los precios que se asemejan unos con otros.

```
[ ]  1 y = d.iloc[:,9] #Información costo.
     2 X = d.iloc[:,10] #Información EPS.
     3
     4 X = np.random.permutation(X.values)
     5 y = np.random.permutation(y.values)
     6 lista = [X,y]
     7 lista = np.array(lista).T
```

Implementación de KMeans.

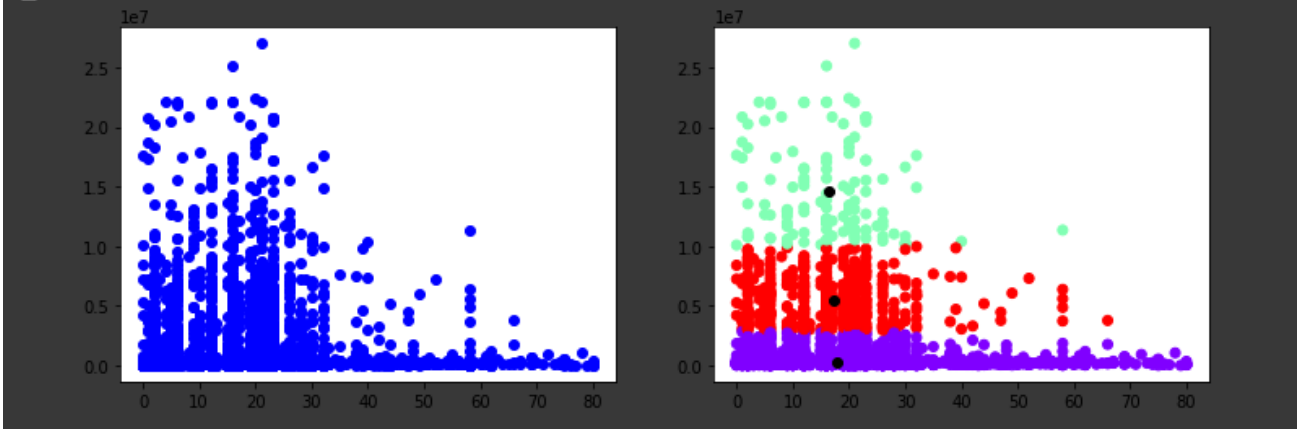
```
[ ]  1 kmeans = KMeans(n_clusters=3)
     2 kmeans.fit(lista)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

Se realizó un entrenamiento con KMans a partir de tres grandes grupos de EPS, es decir, clasificar todas las EPS a partir de los precios y adjuntarlas en tres grandes grupos, esto con el fin de obtener o poder saber en qué grupo podría caer cada EPS a partir del paciente tomado y el precio generado por la atención hospitalaria.

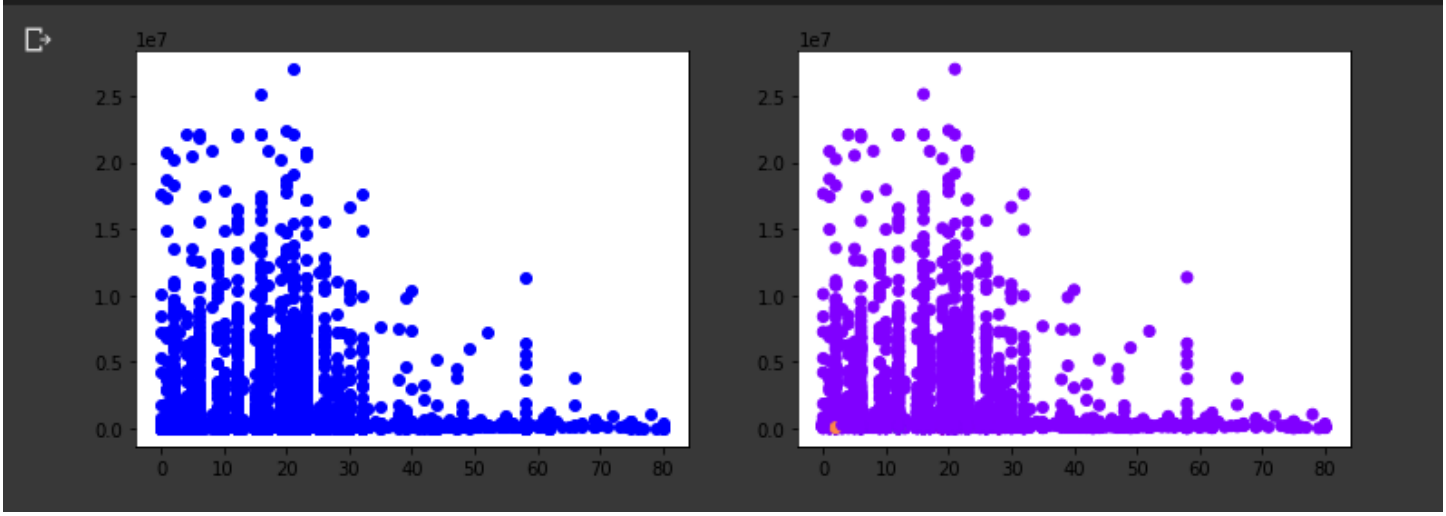
```
[ ]  1 plt.figure(figsize=(12,4))
     2 plt.subplot(121)
     3 plt.scatter(lista[:,0], lista[:,1], c="blue", cmap='rainbow');
     4 plt.subplot(122)
     5 plt.scatter(lista[:,0],lista[:,1], c=kmeans.labels_, cmap='rainbow')
     6 plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

```
<matplotlib.collections.PathCollection at 0x7f9124f02278>
```



En resultado es bastante (gráfica izquierda) respecto a la gráfica derecha. Allí se puede concluir que la toma de los datos está de manera vertical en donde el método KMeans realizó cada procedimiento y enmarcó todos los puntos en cada uno de los clúteres pedidos. No obstante también se trató se realizar el mismo análisis a partir de DBScan.

```
[ ]  1 DBS = DBSCAN(eps=0.1)
     2 DBS.fit(lista)
     3
     4 plt.figure(figsize=(12,4))
     5 plt.subplot(121)
     6 plt.scatter(lista[:,0], lista[:,1], c="blue", cmap='rainbow');
     7 plt.subplot(122)
     8 plt.scatter(lista[:,0],lista[:,1], c=DBS.labels_, cmap='rainbow');
```



DBScan en este caso, ya que es una distribución de puntos ‘orgánica’ tomó dos los puntos como una única clasificación de clúster, por ende allí se concluye que para realizar los clústeres de precios respecto a la EPS el mejor método es KMeans, además se trabajó con diferentes valores de épsilon pero el valor fue el mismo.