PREDICCIÓN DE ACV

BASADO EN FACTORES DE RIESGO

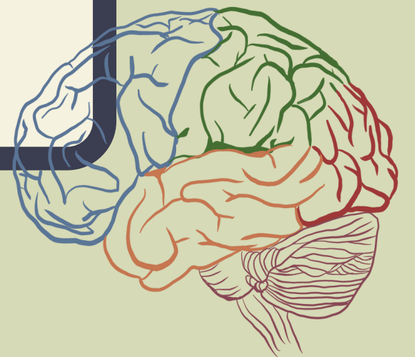
Laura Camila Díaz Delgado -2220100
Daniel Vidal - 2202022
Juan Sebastian Jinete - 2220054

INTRODUCCIÓN

Desarrollar y validar un modelo para predecir la susceptibilidad a accidentes cerebrovasculares (ACV) basado en factores de riesgo conocidos y datos de pacientes. Utilizando técnicas de machine learning & deep learning, se analizarán las características clave de los pacientes para identificar patrones y factores de riesgo asociados con el accidente cerebrovascular. Se explorarán métodos como la clasificación, la regresión y las redes neuronales profundas para mejorar la precisión del modelo.

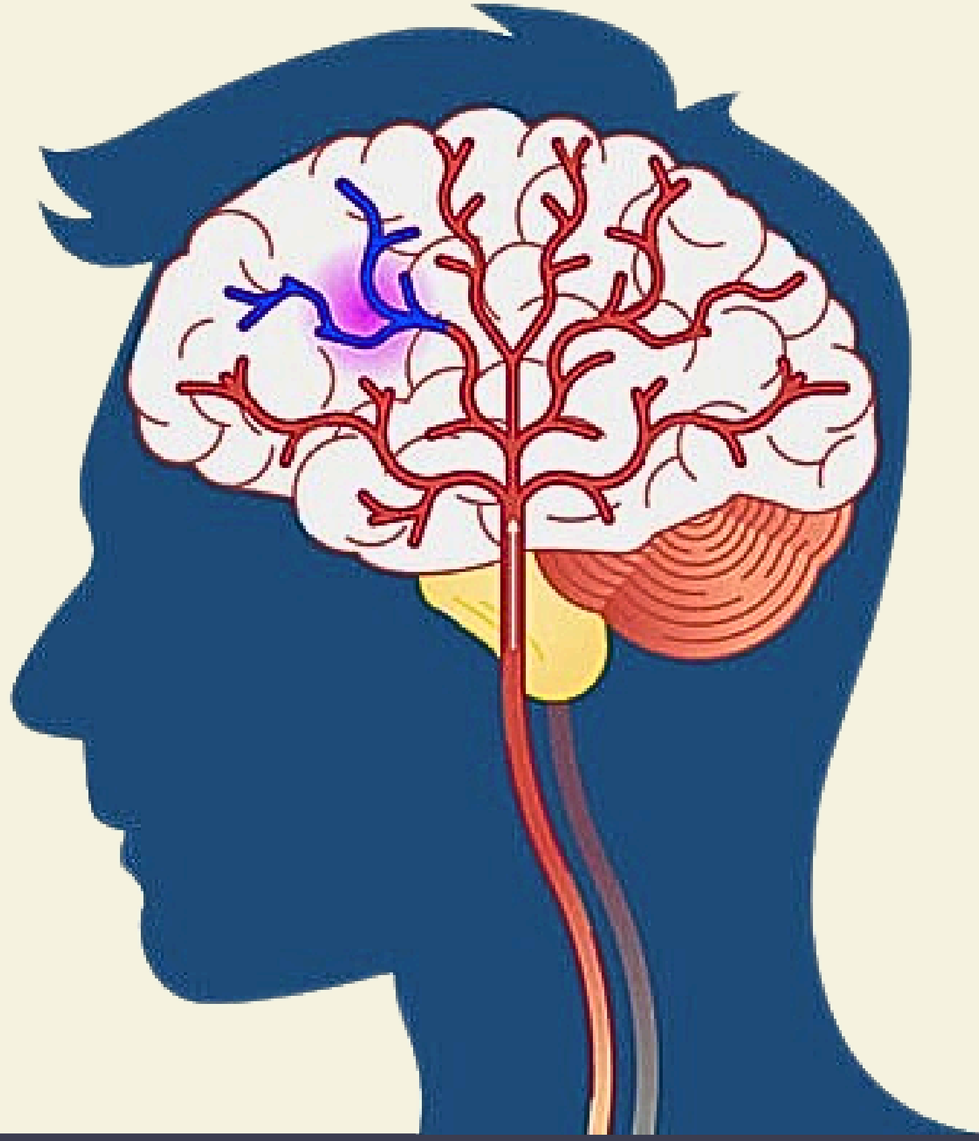
OBJETIVO

Ofrecer una herramienta que permita identificar los riesgos de ACV para que así puedan realizar intervenciones preventivas personalizadas, reduciendo la incidencia y los efectos devastadores de los mismo.



DATASET

Brain
Stroke



DATASET

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-married: "No" or "Yes"
6) work type: "children", "Govtjov", "Never worked", "Private" or "Self-employed"
7) Residencetype: "Rural" or "Urban"
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)

INFORMACIÓN GENERAL

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 43400 entries, 0 to 43399
Data columns (total 12 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   id                  43400 non-null  int64
 1   gender              43400 non-null  object
 2   age                 43400 non-null  float64
 3   hypertension        43400 non-null  int64
 4   heart_disease       43400 non-null  int64
 5   ever_married        43400 non-null  object
 6   work_type           43400 non-null  object
 7   Residence_type      43400 non-null  object
 8   avg_glucose_level   43400 non-null  float64
 9   bmi                 41938 non-null  float64
 10  smoking_status      30108 non-null  object
 11  stroke              43400 non-null  int64
dtypes: float64(3), int64(4), object(5)
memory usage: 4.0+ MB
```

ESTADÍSTICAS

```
df.describe()
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
count	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000	29072.000000
mean	37079.469455	0.614543	47.671746	0.111482	0.052146	0.746079	1.929313	0.497971	106.403225	30.054166	0.969971	0.018850
std	20965.429393	0.487206	18.734490	0.314733	0.222326	0.435261	0.916367	0.500004	45.268512	7.193908	0.676357	0.135997
min	1.000000	0.000000	10.000000	0.000000	0.000000	0.000000	0.000000	0.000000	55.010000	10.100000	0.000000	0.000000
25%	19046.750000	0.000000	32.000000	0.000000	0.000000	0.000000	2.000000	0.000000	77.627500	25.000000	1.000000	0.000000
50%	37444.000000	1.000000	48.000000	0.000000	0.000000	1.000000	2.000000	0.000000	92.130000	28.900000	1.000000	0.000000
75%	55220.250000	1.000000	62.000000	0.000000	0.000000	1.000000	2.000000	1.000000	113.910000	33.900000	1.000000	0.000000
max	72943.000000	2.000000	82.000000	1.000000	1.000000	1.000000	4.000000	1.000000	291.050000	92.000000	2.000000	1.000000

CONTEO DE VALORES FALTANTES

```
valores_faltantes = df.isnull().sum()

(valores_faltantes)
```

```
gender                0
age                   0
hypertension          0
heart_disease         0
ever_married          0
work_type             0
Residence_type        0
avg_glucose_level     0
bmi                   0
smoking_status        0
stroke                0
dtype: int64
```

	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
id	1.000000	-0.010395	0.003718	0.002866	0.012479	0.002221	0.009640	0.000954	0.023062	0.008931	0.002023	-0.000484
gender	-0.010395	1.000000	-0.041219	-0.036955	-0.097699	-0.025376	-0.011566	0.006532	-0.050801	-0.021827	0.012370	-0.012340
age	0.003718	-0.041219	1.000000	0.257564	0.247434	0.547287	0.016762	-0.003554	0.228294	0.106416	-0.152712	0.154060
hypertension	0.002866	-0.036955	0.257564	1.000000	0.117980	0.130813	0.017544	0.002859	0.154063	0.129291	-0.031781	0.078684
heart_disease	0.012479	-0.097699	0.247434	0.117980	1.000000	0.095246	0.032951	0.003118	0.137489	0.022754	-0.035337	0.105149
ever_married	0.002221	-0.025376	0.547287	0.130813	0.095246	1.000000	-0.068288	-0.005055	0.117359	0.143328	-0.055230	0.047738
work_type	0.009640	-0.011566	0.016762	0.017544	0.032951	-0.068288	1.000000	0.009709	0.009850	-0.067224	-0.030287	0.021457
Residence_type	0.000954	0.006532	-0.003554	0.002859	0.003118	-0.005055	0.009709	1.000000	0.002561	0.002852	-0.011928	-0.001967
avg_glucose_level	0.023062	-0.050801	0.228294	0.154063	0.137489	0.117359	0.009850	0.002561	1.000000	0.176897	-0.036799	0.075452
bmi	0.008931	-0.021827	0.106416	0.129291	0.022754	0.143328	-0.067224	0.002852	0.176897	1.000000	-0.036820	-0.004029
smoking_status	0.002023	0.012370	-0.152712	-0.031781	-0.035337	-0.055230	-0.030287	-0.011928	-0.036799	-0.036820	1.000000	-0.019276
stroke	-0.000484	-0.012340	0.154060	0.078684	0.105149	0.047738	0.021457	-0.001967	0.075452	-0.004029	-0.019276	1.000000

No hay correlaciones muy fuertes entre las variables numéricas y la variable objetivo stroke, lo que indica que una combinación de estas características, posiblemente junto con las variables categóricas, podría ser necesaria para predecir eficazmente el riesgo de accidente cerebrovascular. La edad parece tener la correlación más fuerte con stroke, lo cual es consistente con la comprensión médica de que el riesgo de accidente cerebrovascular aumenta con la edad.

RANDOM FOREST

```
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
print("Random Forest Classifier:")
print(confusion_matrix(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classifier:
[[  59 1639]
 [ 124 3993]]
              precision    recall  f1-score   support

           0       0.32      0.03      0.06      1698
           1       0.71      0.97      0.82      4117

    accuracy                           0.70      5815
   macro avg       0.52      0.50      0.44      5815
weighted avg       0.60      0.70      0.60      5815
```

KNEIGHBORSCLASSIFIER

```
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)
print("K-Nearest Neighbors Classifier:")
print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
K-Nearest Neighbors Classifier:
[[ 287 1411]
 [ 611 3506]]
              precision    recall  f1-score   support

           0       0.32      0.17      0.22      1698
           1       0.71      0.85      0.78      4117

    accuracy                           0.65      5815
   macro avg       0.52      0.51      0.50      5815
weighted avg       0.60      0.65      0.61      5815
```

```
svc = SVC(kernel='linear', random_state=42)
svc.fit(X_train, y_train)
y_pred_svc = svc.predict(X_test)
print("Support Vector Machine Classifier:")
print(confusion_matrix(y_test, y_pred_svc))
print(classification_report(y_test, y_pred_svc))
```

```
Support Vector Machine Classifier:
[[   0 1698]
 [   0 4117]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1698
           1       0.71      1.00      0.83      4117

    accuracy                           0.71      5815
   macro avg       0.35      0.50      0.41      5815
weighted avg       0.50      0.71      0.59      5815
```

SUPPORT VECTOR MACHINE

LOGISTIC REGRESSION

```
lr = LogisticRegression(random_state=42)
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)
print("Logistic Regression Classifier:")
print(confusion_matrix(y_test, y_pred_lr))
print(classification_report(y_test, y_pred_lr))
```

```
Logistic Regression Classifier:
[[   0 1698]
 [   0 4117]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1698
           1       0.71      1.00      0.83      4117

    accuracy                           0.71      5815
   macro avg       0.35      0.50      0.41      5815
weighted avg       0.50      0.71      0.59      5815
```

```python
xgb = XGBClassifier(random_state=42)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
print("XGBoost Classifier:")
print(confusion_matrix(y_test, y_pred_xgb))
print(classification_report(y_test, y_pred_xgb))
```

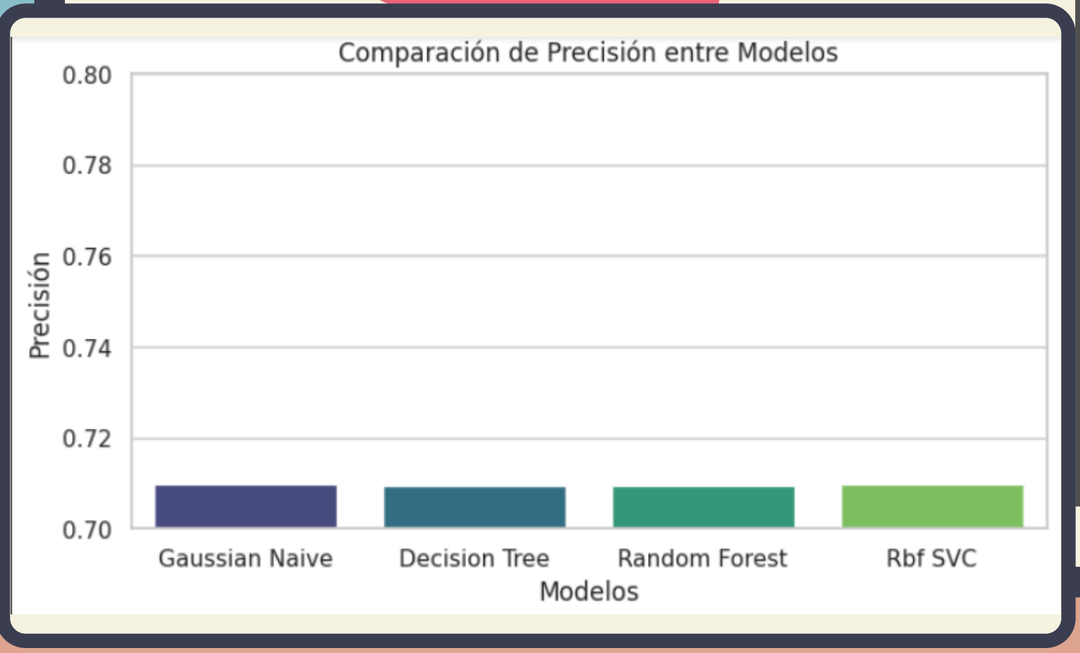
```
XGBoost Classifier:
[[  69 1629]
 [ 173 3944]]
              precision    recall  f1-score   support

           0       0.29      0.04      0.07      1698
           1       0.71      0.96      0.81      4117

    accuracy                           0.69      5815
   macro avg       0.50      0.50      0.44      5815
weighted avg       0.58      0.69      0.60      5815
```

XGBCLASSIFIER

```python
# Definir la arquitectura del modelo
model = Sequential()
model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compilar el modelo
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Entrenar el modelo
history = model.fit(X_train, y_train, epochs=50, batch_size=32, validation_split=0.2, verbose=1)

# Evaluar el modelo
y_pred_nn = model.predict(X_test)
y_pred_nn = (y_pred_nn > 0.5).astype(int)
print("Neural Network Classifier:")
print(confusion_matrix(y_test, y_pred_nn))
print(classification_report(y_test, y_pred_nn))
```

```
Neural Network Classifier:
[[   1 1697]
 [  15 4102]]
              precision    recall  f1-score   support

           0       0.06      0.00      0.00      1698
           1       0.71      1.00      0.83      4117

    accuracy                           0.71      5815
   macro avg       0.38      0.50      0.41      5815
weighted avg       0.52      0.71      0.59      5815
```

NEURAL NETWORK CLASSIFIER




```python
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Evaluar el modelo de Random Forest
scores_rf = cross_val_score(rf, X, y, cv=kf, scoring='accuracy')
print(f"Random Forest CV Accuracy: {scores_rf.mean()}")

# Evaluar el modelo de KNN
scores_knn = cross_val_score(knn, X, y, cv=kf, scoring='accuracy'
print(f"K-Nearest Neighbors CV Accuracy: {scores_knn.mean()}")

# Evaluar el modelo de SVM
scores_svc = cross_val_score(svc, X, y, cv=kf, scoring='accuracy'
print(f"Support Vector Machine CV Accuracy: {scores_svc.mean()}")

# Evaluar el modelo de Regresión Logística
scores_lr = cross_val_score(lr, X, y, cv=kf, scoring='accuracy')
print(f"Logistic Regression CV Accuracy: {scores_lr.mean()}")

# Evaluar el modelo de XGBoost
scores_xgb = cross_val_score(xgb, X, y, cv=kf, scoring='accuracy'
print(f"XGBoost CV Accuracy: {scores_xgb.mean()}")
```

MUCHAS GRACIAS