SISTEMA PREDICTOR DE GRUPOS RELACIONADOS POR EL DIAGNÓSTICO MODELO BASELINE

1st Daniel Palacio Álvarez
Universidad de Antioquia
Medellín, Colombia
daniel.palacioa@udea.edu.co

2rd Cristian Tamayo Arango
Universidad de Antioquia
Medellín, Colombia
cristian.tamayoa@udea.edu.co

3nd Alejandro Agudelo Tamayo
Universidad de Antioquia
Medellín, Colombia
alejandro.agudelot@udea.edu.co

Resumen El proyecto desarrolló un sistema predictivo para clasificar los Grupos Relacionados por el Diagnóstico (GRD) utilizando datos clínicos y administrativos de pacientes hospitalizados. Partiendo de un dataset inicial de 27,869 registros y 68 columnas, se realizó una exploración exhaustiva de los datos, identificando valores nulos, tipos de variables y distribuciones. Se detectó un desbalance en la variable objetivo (GRD-Código), lo que llevó a filtrar las clases con menos de 270 registros, reduciendo el conjunto a 4,626 registros (16.6% del total) y enfocándose en las 10 categorías más representativas. El preprocesamiento incluyó la normalización de variables categóricas y la aplicación de One-Hot Encoding, generando una matriz con 3,715 variables. Para el modelado, se empleó un algoritmo Random Forest con hiperparámetros optimizados (como n_estimators=2000 y class_weight='balanced'), logrando una precisión del 89.2% en el conjunto de prueba. El análisis reveló que el modelo tuvo un alto desempeño en las clases más frecuentes, aunque algunas categorías minoritarias presentaron un recall bajo. Las variables más influyentes incluyeron datos demográficos y clínicos, como edad y tipo de ingreso. Entre las limitaciones destacan la pérdida de datos por eliminación de registros incompletos y el desbalance de clases, sugiriendo futuras mejoras con técnicas como SMOTE o la inclusión de variables temporales. El proyecto demostró la viabilidad de predecir GRD con alta precisión, aportando herramientas valiosas para la gestión hospitalaria de procesamiento.

I. Metodologia

El desarrollo del sistema predictor de Grupos Relacionados por el Diagnóstico (GRD) se implementó en Python utilizando un cuaderno Jupyter Notebook. Inicialmente, se cargó el dataset hospitalario mediante la biblioteca pandas, montando el archivo y leyéndolo en un DataFrame para su posterior exploración

```
import pandas as pd
df = pd.read_csv("archivo_datos.csv")
```

A. Exploracion inicial del Dataset

Para la exploración inicial del dataset con el objetivo de comprender su estructura y dimensiones. Primero, se visualizaron las primeras filas del conjunto de datos mediante la instrucción df.head(), lo que permitió inspeccionar de forma preliminar las variables disponibles, sus nombres, tipos de valores y posibles inconsistencias. Adicionalmente, se determinó el tamaño total del dataset, calculando tanto el número de registros (filas) como la cantidad de variables (columnas) mediante los atributos df.shape[0] y df.shape[1].

Despues se realizó un análisis de la calidad y estructura de los datos, se identificaron los tipos de datos de cada columna mediante el atributo df.dtypes, lo que permitió clasificar las variables según fueran numéricas, categóricas u objetos de texto, se contabilizó cuántas columnas pertenecían a cada tipo de dato utilizando value_counts(), obteniendo así una visión global de la naturaleza del dataset.

B. Valores nulos por columna

Posteriormente, se evaluó la presencia de valores nulos con df.isnull().sum(), filtrando únicamente aquellas columnas que presentaban datos faltantes y ordenándolas de forma descendente para priorizar las más afectadas. Para complementar este análisis, se generó un gráfico de barras horizontales con las 20 columnas que tenían mayor cantidad de valores nulos, utilizando las bibliotecas matplotlib y pandas, lo que facilitó visualizar de manera rápida cuáles variables requerían mayor atención en el preprocesamiento.

C. Estadisticas descriptivas de variables numericas

Para realizar un análisis descriptivo de las variables numéricas del dataset. Se utilizó el método df.describe(include=[np.number]), el cual generó estadísticas básicas como el promedio, la mediana, los valores mínimos y máximos, así como los cuartiles de cada variable cuantitativa. Esto permitió identificar posibles valores atípicos, rangos de variación y tendencias generales en los datos.

De manera complementaria, se analizó específicamente la distribución de la variable "Edad" de los pacientes. Para esto,

se construyó un histograma con 30 intervalos utilizando la biblioteca Seaborn (sns.histplot), incorporando además una curva de densidad (kde) para visualizar la forma de la distribución. Este gráfico permitió observar la frecuencia de pacientes en diferentes rangos etarios y evaluar si la variable presentaba una distribución simétrica, sesgada o multimodal.

D. Distribución de la variable objetivo (GRD -Código)

En esta fase se realizó un análisis de la variable objetivo, que corresponde al código GRD asignado a cada paciente. Primero, se calculó la frecuencia de aparición de cada código mediante df['GRD-Código'].value_counts(), lo que permitió identificar cuántos casos pertenecían a cada categoría. Posteriormente, se extrajo el Top 10 de códigos GRD más frecuentes, mostrando en pantalla las categorías con mayor representación en el conjunto de datos.

Para complementar este análisis, se generó un gráfico de barras horizontales utilizando Seaborn (sns.barplot), donde se visualizaron los diez códigos GRD con mayor número de registros.

E. Relación entre GRD -Código y GRD -Descripción

Sabiendo la distribucion del GRD, se realizó la selección y preparación de las variables relevantes para el modelado predictivo. Primero, se generó una tabla de referencia que relacionaba cada código GRD con su correspondiente descripción, eliminando duplicados y ordenando por código mediante:

```
relacion_grd = df[['GRD-Codigo', 'GRD-Descripcion']]
.drop_duplicates()
.sort_values(by='GRD-Codigo')
```

F. Selección y procesamiento de columnas relevantes

Seguidamente, se definió un subconjunto de columnas útiles para el modelo, incluyendo datos demográficos (Edad, Sexo), administrativos (Aseguradora, Tipo de ingreso), clínicos (Diagnósticos, Procedimientos, Estancia hospitalaria), y la variable objetivo (GRD-Código). Se creó un nuevo DataFrame llamado df_triaje con solo estas variables, lo que redujo la dimensionalidad y eliminó información irrelevante para el problema.Se transformó la columna Fecha de ingreso al formato de fecha estándar (datetime) para extraer nuevas características temporales como el mes de ingreso y el día de la semana de ingreso, las cuales podrían aportar patrones estacionales o de flujo hospitalario. Luego, la fecha original se eliminó para evitar redundancia.

Finalmente, para asegurar la calidad de los datos, se definió un conjunto de columnas obligatorias (Aseguradora, Edad, Sexo, Tipo de ingreso, Diagnóstico de ingreso y Código GRD), eliminando mediante dropna() todos los registros que presentaban valores nulos en estas variables esenciales.

G. Transformacion de datos de fecha y normalizacion de variables categoricas

Se continuó con el preprocesamiento y normalización de las variables para preparar el dataset de manera adecuada para el modelado. Primero, se trabajó con la columna Fecha de ingreso, convirtiéndola al formato estándar de fecha y hora (datetime) mediante pd.to_datetime(). A partir de esta variable se generaron dos nuevas características temporales: el mes de ingreso (.dt.month) y el día de la semana de ingreso (.dt.dayofweek), lo que permitió capturar posibles patrones estacionales o relacionados con la gestión hospitalaria. Posteriormente, la fecha original se eliminó del conjunto (drop(columns='Fecha de ingreso')) para evitar redundancia.

Luego, se definió un conjunto de columnas obligatorias consideradas esenciales para el análisis (Aseguradora, Edad, Sexo, Tipo de ingreso, Diagnóstico de ingreso y Código GRD). Se aplicó dropna(subset=columnas_obligatorias) para eliminar registros con valores faltantes en estas variables clave, asegurando así que la información crítica estuviera completa.

Finalmente, se identificaron todas las columnas categóricas (como Sexo, Tipo de ingreso, Diagnósticos, Procedimientos, Situación al alta, entre otras) y se aplicó una normalización de formato, convirtiendo todos los valores a minúsculas mediante .astype(str).str.lower(). Este paso estandarizó la escritura de las categorías, evitando inconsistencias como diferencias por uso de mayúsculas o variaciones tipográficas que pudieran generar valores duplicados de forma artificial.

H. Eliminación de nulos restantes y filtrado por frecuencia de GRD

Continuando se aplicó una depuración final del dataset para garantizar la calidad y representatividad de las clases antes del modelado, para esto se eliminaron todos los registros que aún presentaban valores nulos en cualquier columna mediante df_triaje.dropna(inplace=True). A continuación, se analizó la distribución de la variable objetivo (GRD -Código), calculando la frecuencia de aparición de cada clase con value_counts(). Dado que algunas categorías tenían un número muy reducido de casos, se decidió filtrar las clases poco representadas para evitar problemas de desbalance extremo que dificultaran el entrenamiento del modelo.

Se estableció un umbral mínimo de 270 registros por clase; las clases que no cumplían con esta frecuencia mínima fueron excluidas. Para ello, se identificaron las clases válidas (clases_validas) y se generó un nuevo subconjunto llamado df_filtrado que incluía únicamente los registros pertenecientes a estas categorías con suficiente representación.

I. Codificación de variables categóricas y división del dataset

En esta etapa se aplicó una codificación One-Hot Encoding a todas las columnas categóricas previamente identificadas. Esto se realizó con pd.get_dummies(), generando variables binarias para cada categoría única y transformando el dataset en un formato completamente numérico compatible con los algoritmos de machine learning.

Luego, se definieron las variables predictoras (X) y la variable objetivo (y). Para ello, se eliminó la columna GRD -Código del conjunto de predictores (X = df_codificado.drop(columns='GRD -Código')) y se conservó como la salida a predecir (y = df_codificado['GRD -Código']).

Posteriormente, se realizó la división del dataset en conjuntos de entrenamiento y prueba utilizando train_test_split de scikit-learn. Se destinó un 20% de los datos para pruebas y un 80% para entrenamiento, aplicando estratificación según la variable objetivo para preservar la misma proporción de clases en ambos conjuntos. Esto garantizó que las clases estuvieran representadas de manera balanceada tanto en la fase de entrenamiento como en la de evaluación.

Se verificaron los tamaños de los conjuntos generados (X_train.shape, X_test.shape) y el número final de clases retenidas tras el filtrado. Finalmente, se calculó el porcentaje de datos utilizados tras las etapas de depuración y filtrado de clases poco representadas. Partiendo de un total de 27.869 registros originales, se cuantificaron los registros válidos en df_filtrado y se obtuvo el porcentaje de retención (porcentaje_retenido), lo que permitió evaluar la reducción del dataset y la cantidad de información que realmente se utilizó para el modelado.

J. Entrenamiento del modelo Random Forest

Para el entrenamiento se utilizo un modelo predictivo utilizando un Random Forest Classifier, algoritmo de ensamble basado en múltiples árboles de decisión.

Para la implementación se empleó la biblioteca scikit-learn, configurando el modelo con una serie de hiperparámetros ajustados para mejorar el rendimiento en un problema multiclase y potencialmente desbalanceado. En particular:

- n_estimators=2000 definió un ensamble de 2000 árboles, aumentando la estabilidad y robustez de las predicciones.
- max_depth=None permitió que los árboles crecieran hasta su máxima profundidad, dejando que el algoritmo determinara automáticamente el nivel óptimo.
- min_samples_split=15 estableció un mínimo de 15 muestras para dividir un nodo, controlando el sobreajuste.
- min_samples_leaf=1 permitió que cada hoja contuviera al menos una muestra.
- max_features='log2' seleccionó un subconjunto reducido de características para cada división, introduciendo mayor diversidad entre los árboles.
- class_weight='balanced' ajustó automáticamente los pesos de las clases en función de su frecuencia, mitigando el problema del desbalance de clases.
- random_state=42 aseguró la reproducibilidad de los resultados.
- n_jobs=-1 habilitó el uso de todos los núcleos de CPU disponibles para paralelizar el entrenamiento y optimizar el tiempo de cómputo.

Finalmente, se entrenó el modelo mediante rf_model.fit(X_train, y_train), utilizando el conjunto de entrenamiento previamente preparado. Con este paso, el modelo Random Forest aprendió los patrones y relaciones entre las variables predictoras y el código GRD, quedando listo para su evaluación en el conjunto de prueba.

K. Evaluación del modelo: Precisión y métricas

Se realizó la evaluación del desempeño del modelo Random Forest entrenado previamente, calculando las precisiones globales tanto en el conjunto de entrenamiento como en el de prueba mediante rf_model.score(), lo que permitió identificar si el modelo presentaba sobreajuste (overfitting) o un desempeño consistente entre ambos conjuntos.

Luego, se generaron las predicciones del modelo sobre los datos de prueba (y_pred = rf_model.predict(X_test)) y se elaboró un classification report utilizando classification_report de scikit-learn. Este reporte incluyó métricas clave para cada clase, tales como:

- Precisión (precision): proporción de predicciones correctas entre todas las predicciones positivas realizadas para una clase.
- Recall (sensibilidad): proporción de verdaderos positivos identificados correctamente respecto a todos los casos reales de la clase.
- F1-score: media armónica entre precisión y recall, útil para evaluar modelos en datasets desbalanceados.
  Support: cantidad de muestras reales por clase en el conjunto de prueba.

El reporte se almacenó en un diccionario (output_dict=True) y posteriormente se convirtió en un DataFrame (df_report) para facilitar su análisis.

Por último, se extrajeron las 10 clases con mayor número de muestras (support) en el conjunto de prueba, ordenándolas de forma descendente. Para estas clases más representativas se mostraron las métricas precision, recall, f1-score y support, redondeadas a dos decimales, permitiendo así evaluar en detalle el rendimiento del modelo en las categorías más relevantes del problema.

L. Matriz de confusión

Con la presicion se realizó un análisis visual de los errores de clasificación del modelo mediante la construcción de una matriz de confusión para las clases más representativas, se identificaron las 10 clases reales más frecuentes en el conjunto de prueba utilizando Counter(y_test).most_common(10). Esto permitió enfocar el análisis en las categorías con mayor cantidad de muestras (Top 10 códigos GRD), que son las más relevantes desde el punto de vista clínico y estadístico.

A continuación, se calculó la matriz de confusión con confusion_matrix(y_test, y_pred, labels=top_labels), restringiendo el análisis a estas clases seleccionadas. La matriz resultante muestra en cada celda el número de registros en los que una clase real fue clasificada como otra, permitiendo identificar tanto las predicciones correctas (diagonal principal) como los errores de confusión entre clases similares.

Continuamente la matriz fue representada como un mapa de calor (heatmap) utilizando la biblioteca Seaborn, con anotaciones numéricas y una escala de color “Reds” para resaltar las frecuencias más altas. Los ejes fueron etiquetados como Real (verdaderos códigos GRD) y Predicción (clases estimadas por el modelo).

M. Top 10 variables más importantes para el modelo

En esta última fase se realizó un análisis de interpretabilidad del modelo Random Forest, enfocándose en identificar las variables más importantes para la predicción del código GRD.

Se obtuvieron las importancias originales de cada característica codificada a partir del atributo rf_model.feature_importances, almacenándolas en una serie de pandas indexada por el nombre de cada variable. Sin embargo, dado que muchas variables categóricas habían sido transformadas mediante One-Hot Encoding, cada categoría generó una nueva columna, lo que fragmentaba la importancia real de la variable original.

Para resolver esto, se implementó una función extraer_columna_original() que tomaba el nombre de cada columna y eliminaba el sufijo creado por la codificación (por ejemplo, "Sexo_masculino" se agrupaba como "Sexo"). Luego, se agruparon todas las categorías pertenecientes a la misma variable original y se sumaron sus importancias, obteniendo así una visión más fiel del peso real de cada variable antes de la codificación.

A partir de esta agregación, se ordenaron las variables por su relevancia y se seleccionaron las 10 más importantes para la predicción del modelo. Finalmente, se generó un gráfico de barras horizontales con Seaborn mostrando el Top 10 de características más influyentes, destacando visualmente aquellas variables originales que más contribuyen a la clasificación de los pacientes en los distintos códigos GRD.

II. Analisis de los resultados

A. Procesamiento de la base de datos

En el inicio se lleva a cabo una exploración detallada del dataset. Se identifica que el conjunto de datos cuenta con 27.869 registros y 68 columnas, lo que sugiere un volumen considerable de información para el modelado. Al revisar los tipos de datos, se observa que existe una mezcla de variables numéricas, categóricas y de texto, con un predominio de 31 columnas de tipo numérico, 27 de tipo objeto (cadenas de texto) y 10 enteras. Esto permite inferir que se requerirá un tratamiento diferenciado para cada tipo de variable, especialmente en el proceso de codificación y estandarización.

Uno de los primeros hallazgos importantes es la presencia de valores nulos en varias columnas Figura 1, sobre todo en aquellas relacionadas con procedimientos médicos, como Proc27, Proc28, Proc29 y Proc30, que muestran una gran proporción de datos faltantes. Además, se muestran las primeras filas del dataset, lo que permite identificar ejemplos de aseguradoras, grupos de edad, tipos de ingreso y descripciones clínicas que acompañan al paciente en cada registro.

Respecto a la variable objetivo, denominada GRD - Código, se presenta su distribución, evidenciando que algunos códigos son mucho más frecuentes que otros Figura 2., lo que revela un posible desbalance de clases. lo cual es importante conocer pues condiciona la estrategia de entrenamiento del modelo, dado que los grupos mayoritarios podrían sesgar las predicciones

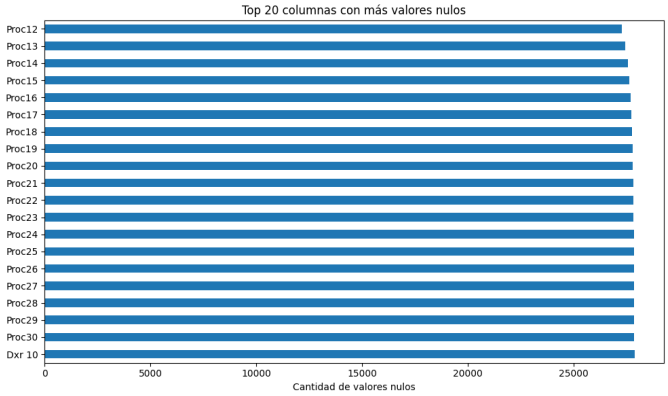

Fig. 1: Cantidad de valores nulos por columna

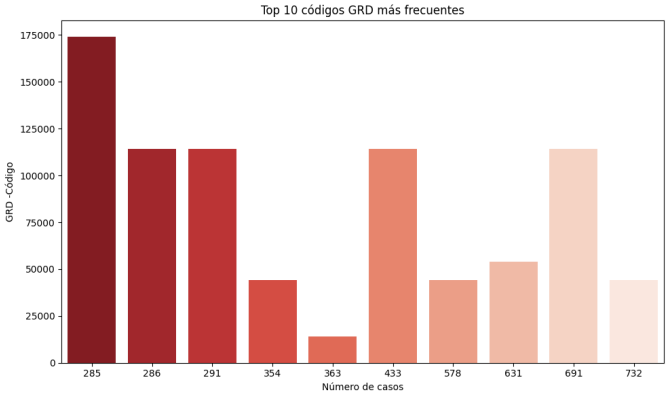

Fig. 2: Distribuicion de los GRD

También se realiza un mapeo entre el código GRD y su respectiva descripción, permitiendo contextualizar qué significa cada número en términos clínicos, impotante para la interpretación final del modelo, ya que un mismo código puede representar un diagnóstico complejo, por ejemplo "Infección por virus de inmunodeficiencia humana con complicaciones" o "Accidente cerebrovascular con infarto".

Otro aspecto es el tratamiento de variables temporales, donde se transforma la fecha de ingreso de texto a formato datetime, permitiendo extraer nuevas variables como el mes

	GRD -Código	GRD -Descripción
6067	11011	PH PROCEDIMIENTOS VASCULARES INTRACRANEALES
168	11012	PH PROCEDIMIENTOS VASCULARES INTRACRANEALES w/CC
71	11013	PH PROCEDIMIENTOS VASCULARES INTRACRANEALES w/MCC
1393	11101	PH CRANEOTOMÍA
88	11102	PH CRANEOTOMÍA w/CC
13	11103	PH CRANEOTOMÍA w/MCC
27200	11111	PH PROCEDIMIENTOS DE DERIVACIÓN VENTRICULAR
1456	11112	PH PROCEDIMIENTOS DE DERIVACIÓN VENTRICULAR w/CC
3311	11113	PH PROCEDIMIENTOS DE DERIVACIÓN VENTRICULAR w/MCC
6259	11121	PH PROCEDIMIENTOS VASCULARES EXTRACRANEALES

Fig. 3: Descripcion de los GRD

y el día de la semana. Continuando con esto se realiza una depuración más estricta de los datos, eliminando registros que carecían de información clave como aseguradora, edad, sexo, tipo de ingreso, diagnóstico o código GRD. Asimismo, se filtran las clases GRD con muy pocas muestras, estableciendo un umbral mínimo de 270 registros por clase. Como consecuencia, de los 27.869 registros originales, se retienen únicamente 4.626 registros (alrededor del 16,6% del total), concentrándose en las 10 clases más representativas.

Posteriormente, se normalizan las variables categóricas para evitar inconsistencias en los nombres (por ejemplo, se pasan a minúsculas) y se codifican mediante One-Hot Encoding, lo que genera una matriz de entrenamiento con 3.715 variables. Esta transformación permite que el modelo de Machine Learning procese adecuadamente la información categórica. Luego se divide el dataset en conjunto de entrenamiento y prueba de forma estratificada para mantener la proporción de clases, resultando en 3.700 registros para entrenamiento y 926 para prueba.

B. Modelado

Para el modelado se selecciona un Random Forest, un algoritmo robusto para clasificación multiclase. Se entrena utilizando los hiperparametros mencionados en la metodologia los cuales fueron definidos previamente en el proyecto anterior, donde se realizó una búsqueda mediante la función de búsqueda de hiperparámetros de scikit-learn (GridSearchCV). En esa etapa previa se evaluaron múltiples combinaciones de parámetros como el número de árboles (n_estimators), el criterio de división, la profundidad máxima y el número mínimo de muestras por división. Tras esta optimización extendida se determinaron los valores más adecuados, como n_estimators=2000, class_weight='balanced', max_features='log2' y min_samples_split=15, que fueron directamente reutilizados en este proyecto para garantizar el mejor desempeño posible sin necesidad de recalibrar el modelo desde cero.

Durante la evaluación, el modelo alcanza una precisión en entrenamiento de 99,97%, lo que confirma que aprende bien los patrones, y una precisión en prueba de 89,2%, un desempeño muy alto considerando la complejidad multiclase del problema.

Las métricas de evaluación por clase muestran que algunas categorías tienen precision y recall superiores al 95%, especialmente aquellas más frecuentes, mientras que otras con menos datos presentan recall más bajo, como el caso del GRD 114123 con solo 58% de recall, evidenciando que la frecuencia de clase sigue influyendo en la calidad de predicción.

Finalmente para el desarrollo de la aplicacion en django es clave es el análisis de las 10 variables más importantes para el modelo, extraídas de las importancias del Random Forest. Estas variables, posiblemente relacionadas con edad, tipo de ingreso, aseguradora o diagnósticos previos, son las que más contribuyen a la decisión final del clasificador.

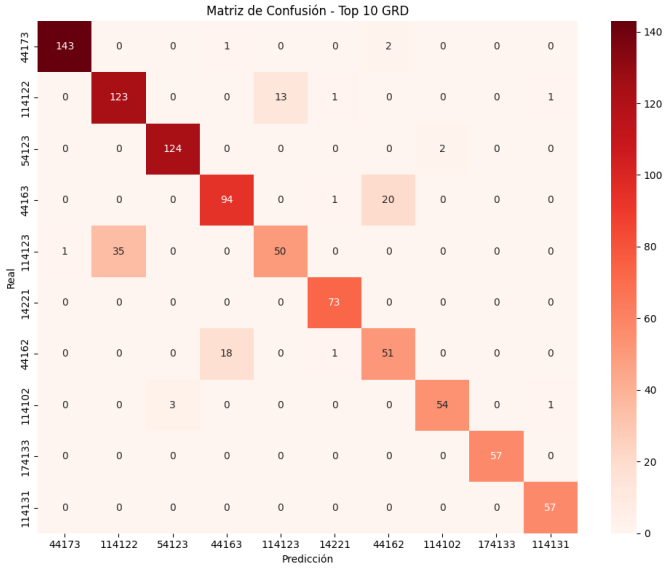

Fig. 4: Matriz de confusion

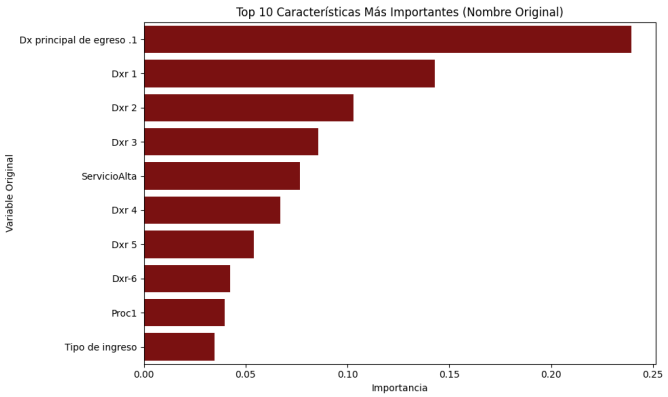

Fig. 5: Caracteristicas mas importantes para el modelo

III. Conclusiones

En este proyecto se construyó un sistema predictivo capaz de clasificar correctamente los Grupos Relacionados por el Diagnóstico (GRD) con base en datos clínicos y administrativos de pacientes hospitalizados.

1) Principales hallazgos:

- Se realizó una limpieza cuidadosa del dataset, eliminando registros con valores nulos en variables críticas y filtrando clases con baja representación.
- Se hizo una exploración estadística descriptiva (EDA), con análisis de tipos de datos, distribución de variables numéricas, identificación de valores nulos y visualización de la variable objetivo.
- El modelo Random Forest alcanzó una **precisión en prueba del 89.2%**, lo cual es bastante alto para una clasificación multiclase.
- La matriz de confusión y el análisis por clase mostraron que el modelo tiene un buen desempeño en las clases

más frecuentes, aunque hay clases con menor recall que podrían ser analizadas más a fondo.
- Las variables más influyentes para el modelo fueron [reemplazar aquí con las que te salieron en tu gráfica], lo cual permite una mejor interpretación de los factores que contribuyen a la clasificación GRD.

2) Limitaciones y recomendaciones:

- El dataset original contenía muchas columnas con valores faltantes o irrelevantes. En futuros trabajos se podría considerar imputar algunos valores en lugar de eliminarlos para conservar más datos.
- Algunas clases GRD tienen muy poca representación y fueron excluidas del análisis. Podría explorarse el uso de técnicas de balanceo como SMOTE o submuestreo.
- El modelo Random Forest funcionó muy bien, pero sería interesante comparar su desempeño con otros modelos como Gradient Boosting o XGBoost.
- No se incluyeron aún variables temporales como estacionalidad, que podrían enriquecer el modelo en análisis futuros.