Challenge DataIQ - Informe

A continuación se presenta un informe que detalla el proceso de trabajo llevado a cabo para elaborar un clasificador que permite predecir (con cierta precisión) si una parsona será internada o no al llegar a la guardia de un hospital.

Se ha realizado un analisis exploratorio de los datos para indentificar posibles correlaciones entre la variable objetivo y los features presentes, tantos los que se encontraban desde el inicio como también algunos generados a partir de la misma observación y de cierta intuicion (luego revisada) para encontrar así dónde conviene enfocar los esfuerzos por construir el mejor clasificador posible.

Nota 1:

- Las funciones de preprocesamiento de los datos se encuentran en el archivo "preprocessing.py"
- Las funciones de modelado se encuentran en el archivo "modeling.py"
- Las funciones de evaluacion del modelo y presentacion de métricas se encuentran en el archivo "evaluation.py"

Nota 2:

- Hay un archivo llamado "main_hospital_classifier.py" que realiza el mismo procedimiento que esta notebook
- Hay un archivo llamado "model_log.csv" que guarda los resultados y los datos de cada mejor modelo evaluado(*1)
- Cada mejor modelo evaluado se guarda automaticamente con un nombre de referencia que se correlaciona con una entrada en el archivo "model_log.csv"

Definir variables de configuracion

```
from dataclasses import dataclass
from ydata_profiling import ProfileReport
from src.preprocessing import load_and_read_data
from src.preprocessing import crear_tipo_episodio
from src.preprocessing import default_clean
from src.preprocessing import create_final_dataframe
from src.preprocessing import check_final_data
from src.preprocessing import prepare_test_train
from src.modeling import evaluate_models
from src.modeling import train_and_evaluate_model
from src.evaluation import full_metrics_eval
from src.evaluation import save_model_with_metadata


## Definicion de variables a utilizar
@dataclass
class Config:

    archivos=['Episodios_Diagnosticos.csv',     # Nombre de los archivos que contienen la información a procesar
              'Estudios_Complementarios.csv',
              'Pacientes.csv',
              'Signos_Vitales.csv']

    path = './data/'    # Ruta donde se encuentran los archivos

    n_charenc=10000   # cantidad de caracteres para evaluar el encoding de los archivos

    features=['EDAD',                     # features a considerar para el modelo (chequear que los nombres correspondan a features en df_fina
              'SEXO',
              'CANTIDAD_EPISODIOS',
              'CANTIDAD_ESTUDIOS',
              'CANTIDAD_SIGNOS_VITALES',
              'PRIMER_AREA_FRECUENTE',
              'ULTIMO_AREA_FRECUENTE',
              'TIPO_DIAGNOSTICO_FRECUENTE'
             ]

    test_size=0.20     # proporcion del dataset de testing
    seed=42            # semilla de randomizacion

    model="Random Forest"  # modelo a utilizar en el entranmiento y evaluacion
    cv_split=10            # segmentacion para el proceso de cross-validation
```

```
config=Config()
```

Preprocesamiento de Data - Análisis Exploratorio

```
# Evaluacion, lectura y carga de datos

dfs = load_and_read_data(file_names=config.archivos, path=config.path, n_charenc=config.n_charenc)
```

```
Successfully read Episodios_Diagnosticos.csv with encoding: ISO-8859-1
Warning: Failed to read Estudios_Complementarios.csv with detected encoding (ascii). Error: 'ascii' codec can't decode byte 0xf3 in posi
Successfully read Estudios_Complementarios.csv with encoding: ISO-8859-1
Successfully read Pacientes.csv with encoding: ISO-8859-1
Warning: Failed to read Signos_Vitales.csv with detected encoding (ascii). Error: 'ascii' codec can't decode byte 0xd0 in position 76541
Successfully read Signos_Vitales.csv with encoding: ISO-8859-1
Total DataFrames loaded: 4
```

```
# Division de dataframes para cada archivo

df_episodios = dfs[0]
df_estudios = dfs[1]
df_pacientes = dfs[2]
df_signos = dfs[3]


profile = ProfileReport(df_episodios, title="Reporte de Episodios")
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Reporte de Episodios

Overview Variables Interactions Correlations Missing values Sample Duplicate rows

Correlations

Auto

Heatmap Table

	ID_INTERNACION	PACIENTE	PRIMER_AREA	TIPO_DIAGNOSTICO	TIPO_EPISODIO	ULTIMO
ID_INTERNACION	1.000	0.057	0.181	0.145	0.276	0.190
PACIENTE	0.057	1.000	0.147	0.041	0.267	0.151
PRIMER_AREA	0.181	0.147	1.000	0.189	0.492	0.949
TIPO_DIAGNOSTICO	0.145	0.041	0.189	1.000	0.363	0.203
TIPO_EPISODIO	0.276	0.267	0.492	0.363	1.000	0.494
ULTIMO_AREA	0.190	0.151	0.949	0.203	0.494	1.000

Missing values

Algunas observaciones sobre df_episodios:

- Se observa que primera y utlima area están altamente correlacionadas.
- Se observa también que "Clinica Medica" y "Obstetricia" acumulan el 73% de las areas en las que se dan los episodios
- Se observa que el lugar donde se encuentra la variable target (tipo episodio) se encuentra más correlacionada tanto con primera como con ultima area

```
# Agregar 'CLASE' al df de Episodios en base a 'TIPO_EPISODIO' ('CLASE'=1 si 'TIPO_EPISODIO' = 'H',  'CLASE' = 0 en otro caso)
df_episodios = crear_tipo_episodio(df_episodios)
```

```
df_episodios.dtypes
```

```
PACIENTE                    int64
ID_INTERNACION              int64
FECHA_Y_HORA_DE_INGRESO    object
FECHA_HORA_EGRESO_FISICO   object
PRIMER_AREA                object
ULTIMO_AREA                object
RAZON_INTERNACION          object
TIPO_DIAGNOSTICO           object
DESCRIPCION                object
TIPO_EPISODIO              object
CLASE                       int64
dtype: object
```

```
# Limpieza inicial, especificada en la documentación: Eliminacion de registros con 'TIPO_EPISODIO'='*' en df_episodios,
#                                                     Eliminación de registros ducplicados en df_pacientes,
#                                                     Eliminacion de columno 'ID_ITEM' en df_estudios.
df_episodios, df_pacientes, df_estudios = default_clean(df_episodios, df_pacientes, df_estudios)
```

Ahora que sea ha agregado la variable 'CLASE' y se ha limpiado el dataframe, se observa nuevamente el informe del mismo:

```
profile = ProfileReport(df_episodios, title="Reporte de Episodios")
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Reporte de Episodios

Overview Variables Interactions Correlations Missing values Sample Duplicate rows

Correlations

Auto

Heatmap Table

	CLASE	ID_INTERNACION	PACIENTE	PRIMER_AREA	TIPO_DIAGNOSTICO	TIPO_EPISODIO
CLASE	1.000	0.465	0.241	0.432	0.736	1.000
ID_INTERNACION	0.465	1.000	0.179	0.179	0.131	0.279
PACIENTE	0.241	0.179	1.000	0.154	0.066	0.235
PRIMER_AREA	0.432	0.179	0.154	1.000	0.175	0.545
TIPO_DIAGNOSTICO	0.736	0.131	0.066	0.175	1.000	0.409
TIPO_EPISODIO	1.000	0.279	0.235	0.545	0.409	1.000
ULTIMO_AREA	0.460	0.188	0.161	0.980	0.199	0.551

Se observa que existe una elevada correlacion entre 'CLASE' con 'TIPO_DIAGNOSTICO' y, como era de esperarse, con ambas areas, última y primera.

Se toman entonces decisiones sobre qué datos son importantes para realizar la clasificacion:

- Se agregará por paciente el 'TIPO_DIAGNOSTICO' más frecuente
- Se agregará por paciente el 'PRIMER_AREA' más frecuente
- Se agregará por paciente el 'ULTIMO_AREA' más frecuente

Tambien por intuicion, se asume que es probable que exista una correlacion entre la cantidad de estudios realizados y la posibilidad de internacion Si alguien se hizo muchos estudios, tanto de signos vitales como de estudios complementarios, es probable que alguno(s) de ellos haya(n) sido durante una internación. Entonces tambien:

- Se agregará por paciente la cantidad de Estudios Complementarios realizados (se cuenta la cantidad de campos por paciente en df_estudios)
- Se agregará por paciente la cantidad de Signos Vitales medidos (se cuenta la cantidad de campos por paciente en df_signos)

Así se crea el DataFrame df_final, a continuación:

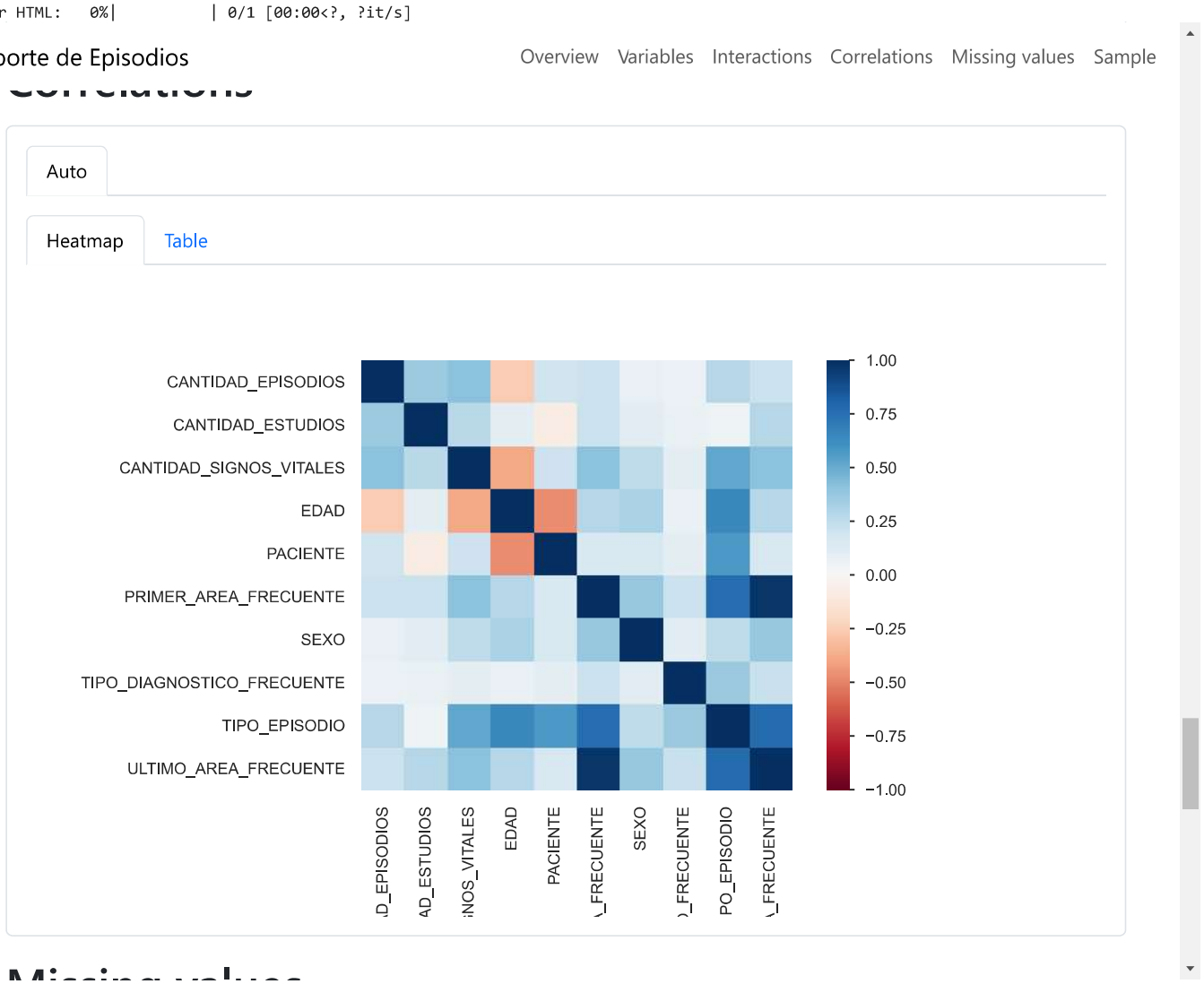
```
# Creacion del DataFrame final, que será el dataset que se dividirá en entrenamiento y testing. Consultar documentacion aparte sobre que cri
# 'create_final_dataframe()' en 'preprocessing.py' contiene ena explicacion parcial de la misma
df_final = create_final_dataframe(df_pacientes, df_episodios, df_signos, df_estudios)


# Revisión de la longitud del dataset. Dado que los registros que se utilizaran corresponden a pacientes,
#  el df_final debe ser de la misma longitud que df_pacientes, ya que la informacion concatenada en el mismo es relativa a los pacientes.
```

```
check_final_data(df_final,df_pacientes)
```

```
La cantidad de registros es la correcta: total de  2635
```

```
profile = ProfileReport(df_final, title="Reporte de Episodios")
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```



Algunas observaciones sobre el df_final:

- Hay una gran mayor proporción de mujeres (F) que de hombres (M) entre los pacientes. Ésto podria haberse observado directamente en un reporte sobre los pacientes a partir de df_pacientes, pero ésta distinción toma otra relevancia ahora que también se sabe que una de las areas de mayor tránsito es Obstetricia.
- La variable 'TIPO_EPISODIO' ahora es una variable binaria que contiene 1 si es internado y 0 si no es.
- Hay casi el doble de pacientes NO internados ('TIPO_EPISODIO'=0) que internados ('TIPO_EPISODIO'=1), con lo cual el dataset se encuentra desbalanceado en el target
- Se observa que 'TIPO_EPISODIO' mantiene la correlacion con las areas (obviamente), pero también presenta correlación con EDAD, con Cantindad de estudios de signos vitales, y en menos medida con diagnostico_frecuente y con cantidad_episodios

Se construirá el clasificador a partir de los features basados en éstas variables.

Observamos más claramente los tipos de datos en df_final:

```
df_final.dtypes
```

```
PACIENTE                      int64
EDAD                          int64
SEXO                         object
CANTIDAD_EPISODIOS          float64
CANTIDAD_ESTUDIOS             int64
CANTIDAD_SIGNOS_VITALES       int64
PRIMER_AREA_FRECUENTE        object
ULTIMO_AREA_FRECUENTE        object
TIPO_DIAGNOSTICO_FRECUENTE   object
TIPO_EPISODIO                 int32
dtype: object
```

Dado que varios de los campos de df_final son datos categóricos, es conveniente realizar una tranformación de tipo OneHotEncoding para dichas variables. A continuación se realiza eso mismo y se preparan los datos para las etapas de entrenemiento y de test.

Tambien se seleccionan los features detallados en el archivo de configuración (mas arriba)

```
# Extraccion de los conjuntos de entrenamiento y testeo. Se seleccionan los features, la proporcion del set de test y
# la semilla de randomizacion (todas en el archivo de configuracion)
X_train, X_test, y_train, y_test = prepare_test_train(df_final,features=config.features, test_size=config.test_size, seed=config.seed)
```

Entrenamiento y evaluacion del modelo

```
# Entrenamiento y evaluacion en una serie de modelos posibles. Eleccion del mejor modelo (en base unicamente a la precision)
df_evaluate_models, best_model = evaluate_models(X_train, X_test, y_train, y_test, test_size=config.test_size, random_state=config.seed)
```

```
                    Model  Accuracy  Precision    Recall  F1-Score       AUC
1           Random Forest  0.912713   0.900552  0.853403  0.876344  0.962766
6       Gradient Boosting  0.912713   0.896175  0.858639  0.877005  0.960865
0     Logistic Regression  0.910816   0.900000  0.848168  0.873315  0.951306
3           Decision Tree  0.895636   0.857895  0.853403  0.855643  0.886523
5             Naive Bayes  0.886148   0.832487  0.858639  0.845361  0.935007
4     K-Nearest Neighbors  0.850095   0.804348  0.774869  0.789333  0.885845
2  Support Vector Machine  0.842505   0.772727  0.801047  0.786632  0.884271
Best performance model in Accuracy is:  Random Forest
```

```
df_evaluate_models
```

	Model	Accuracy	Precision	Recall	F1-Score	AUC
1	Random Forest	0.912713	0.900552	0.853403	0.876344	0.962766
6	Gradient Boosting	0.912713	0.896175	0.858639	0.877005	0.960865
0	Logistic Regression	0.910816	0.900000	0.848168	0.873315	0.951306
3	Decision Tree	0.895636	0.857895	0.853403	0.855643	0.886523
5	Naive Bayes	0.886148	0.832487	0.858639	0.845361	0.935007
4	K-Nearest Neighbors	0.850095	0.804348	0.774869	0.789333	0.885845
2	Support Vector Machine	0.842505	0.772727	0.801047	0.786632	0.884271

```
# Agrego el mejor modelo al archivo de configuracion
config.model = best_model
print(config.model)
```

```
Random Forest
```

```
print("El mejor modelo es:",config.model)
```

```
El mejor modelo es: Random Forest
```

```
# Dado el mejor modelo en la evaluacion, re-entreno el mismo y lo re-evaluo

model, accuracy, precision, recall, f1, auc = train_and_evaluate_model(X_train, X_test, y_train, y_test, model=config.model, seed=config.see
#model, accuracy=train_and_evaluate_model(X_train, X_test, y_train, y_test, model="Gradient Boosting", seed=config.seed)
```

```
Modelo utilizado:  Random Forest
Exactitud del modelo: 0.91
```

OBS: Se podria haber obtenido directamente el objeto modelo a partir de 'evaluate_models()',

pero he optado por realizar ésta segunda accion por separado en base al siguiente criterio:

1) En esta instancia no es relevante optimizar la eficiencia 2) El tamaño del dataset, y por lo tanto tambien el tiempo de entrenamiento de los modelos evaluados, no significan un consumo elevado de recursos en la instancia actual de devlopment (sí tendría un impacto en una posible instancia de produccion). 3) En pos de la claridad del codigo en esta instancia resulta preferible no anidar demasiadas funciones, por lo tanto prefiero no reutilizar 'train_and_evaluate_model()' dentro de 'evaluate_models()'

```
# Evaluacion de metricas tipicas para el modelo entrenado: cross-validation, matriz de confusion, recall, F1
cv_score, y_pred, cm, cr=full_metrics_eval(model, X_train, y_train, X_test, y_test, cv_split=config.cv_split)
```

```
Promedio de los puntajes de validación cruzada: 0.92
Matriz de Confusión:
[[318  18]
 [ 28 163]]
Matriz de confusión porcentual:
[[94.64  5.36]
 [14.66 85.34]]
Reporte de Clasificación:
              precision    recall  f1-score   support

           0       0.92      0.95      0.93       336
           1       0.90      0.85      0.88       191

    accuracy                           0.91       527
   macro avg       0.91      0.90      0.90       527
weighted avg       0.91      0.91      0.91       527
```

Algunas observaciones particulares para éstos resultados:

- El valor de cross-validation es consistente con el valor de precisión.
- El modelo es bastante bueno para predecir a los NO internados, pero empeora un poco para la predicción de los internados. Ésto se podría justificar a partir del desbalance de los datos en la proporcion (H / ~H)
- Sobre el punto anterior me extiendo en las observaciones finales.

Algunas observaciones finales sobre éstos resultados:

- Se ha utilizado OneHotEncoding para la representacion de datos categoricos, con lo cual hace sentido que un clasificador del tipo "Random Forest" se ajuste bien, teniendo en cuenta la división en regiones no solapantes que realiza el mismo.
- Se ha experimentado con otras combinaciones de features (apagando algunos y encendiendo otros), y por ahora el modelo que mejor performance ha tenido en términos de precición y en términos generales ha sido Random Forest utilizando todos los features presentes.
- Es importante notar que se cuenta con un dataset inicial desbalanceado, es decir, hay muchos menos casos de internación que de no internación. También es preciso considerar la importancia que le da el hospital a un error de prediccion de tipo II, es decir, si un paciente que debe ser internado no se lo interna (un verdadero clasificado como falso). En ese caso tal vez sería necesario encarar el análisis ponderando más el valor de Recall que el de precisión. De nuevo, ésto tendría que verse en el caso de que el hospital, por algún motivo, le convenga (o no) optimizar la predicción en algún sentido particular.
- Se podría profundizar el analisis buscando si existe algún patron entre las muestras mal clasficadas, pero sería prudente conocer si hay objetivos más específicos para el clasificador antes de invertir energía en dicho esfuerzo.

```
metrics = {
    "accuracy": accuracy,
    "precision": precision,
    "recall": recall,
    "f1": f1,
    "auc": auc
}
```

```
save_model_with_metadata(model, config.model, metrics, cv_score, cm, cr, config, "models", "model_log.csv")
```

```
Modelo y metadatos guardados en: models\Random Forest_0.91_0.85_20241220_110237.pkl
Registro actualizado en: model_log.csv
```

```
'models\\Random Forest_0.91_0.85_20241220_110237.pkl'
```

```
#import numpy as np

#cm_percent = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis] * 100
#print("Matriz de confusión porcentual:")
#print(np.round(cm_percent, 1))
```

Ajuste de Hiperparámetros

Dado que el mejor modelo encontrado en este caso es un clasificador "Random Forest" queda entonces encontrar el mejor ajuste por hiperparámetros para el mismo:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
```

```
rf = RandomForestClassifier(random_state=42)
```

```
# Definir los hiperparámetros a probar
param_grid = {
    'n_estimators': [100, 200, 300],         # Número de árboles en el bosque
    'max_depth': [10, 20, 30, None],        # Profundidad máxima del árbol
    'min_samples_split': [2, 5, 10],        # Mínimas muestras para dividir un nodo
    'min_samples_leaf': [1, 2, 4],          # Mínimas muestras en cada hoja
    'max_features': ['sqrt', 'log2', None]  # Máximas características consideradas en cada división
}
```

```
# Configurar el GridSearchCV
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    cv=5,                   # Validación cruzada de 5 divisiones
    scoring='accuracy',     # Métrica obj
    verbose=2,              # Que detalle el progreso
    n_jobs=-1               # Usar todos los núcleos
)
```

```
# Entrenar el modelo con los parámetros óptimos
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 324 candidates, totalling 1620 fits
```

▸ GridSearchCV

▸ estimator: RandomForestClassifier

▸ RandomForestClassifier

```
# Imprimir los mejores parámetros y el mejor score
print("Mejores hiperparámetros (segun accuracy):", grid_search.best_params_)
print(f"Mejor puntuación de validación cruzada: {grid_search.best_score_:.2f}")
```

```
Mejores hiperparámetros (segun accuracy): {'max_depth': 20, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_est
Mejor puntuación de validación cruzada: 0.93
```

```
# Evaluar el modelo en los datos de prueba
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
```

```
print("Reporte de clasificación:")
print(classification_report(y_test, y_pred))
```

```
Reporte de clasificación:
              precision    recall  f1-score   support

           0       0.92      0.94      0.93       336
```

```
                1       0.90      0.86      0.88       191

         accuracy                           0.91       527
        macro avg       0.91      0.90      0.91       527
     weighted avg       0.91      0.91      0.91       527
```

```
print("Matriz de confusión:")
print(confusion_matrix(y_test, y_pred))
```

```
Matriz de confusión:
[[317  19]
 [ 26 165]]
```

```
save_model_with_metadata(model, config.model, metrics, cv_score, cm, cr, config, "models", "model_log.csv")
```

```
Modelo y metadatos guardados en: models\Random Forest_0.91_0.85_20241220_112252.pkl
Registro actualizado en: model_log.csv
'models\\Random Forest_0.91_0.85_20241220_112252.pkl'
```