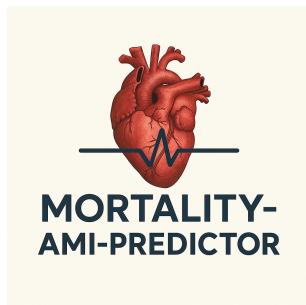




Informe Técnico

Mortality AMI Predictor

Sistema de Predicción de Mortalidad Intrahospitalaria
en Pacientes con Infarto Agudo de Miocardio



Versión 1.0

Integrantes del Equipo:

- Richard Alejandro Matos Arderí
- Abel Ponce González
- Abraham Romero Imbert

- Michell Viu Ramirez
- Eveliz Espinaco Milián
- Eduardo Brito Labrada
- Ernesto Abreu Peraza

Institución:

Facultad de Matemática y Computación
Universidad de La Habana

Registro Cubano de Infarto Agudo del Miocardio

Tecnologías: Python · Streamlit · Scikit-learn · XGBoost · SHAP

Arquitectura: Modular · Orientada a Objetos · Patrones de Diseño

Índice

1 Resumen Ejecutivo	4
1.1 Características Principales	4
1.2 Métricas de Calidad del Código Fuente	5
2 Arquitectura del Sistema	5
2.1 Visión General de la Arquitectura	5
2.2 Estructura de Directorios	5
2.3 Patrones de Diseño Implementados	6
3 Tecnologías y Dependencias	7
3.1 Stack Tecnológico Principal	7
3.2 Dependencias Principales	7
4 Módulos Funcionales	8
4.1 Módulo de Carga de Datos (<code>data_load</code>)	8
4.1.1 Componentes	8
4.1.2 Características Técnicas	8
4.2 Módulo de Limpieza de Datos (<code>cleaning</code>)	9
4.2.1 Componentes	9
4.2.2 Configuración de Limpieza	10
4.3 Módulo de Análisis Exploratorio (<code>eda</code>)	11
4.3.1 Componentes	11
4.3.2 Estadísticas Computadas	11
4.4 Módulo de Características (<code>features</code>)	12
4.4.1 Componentes	12
4.4.2 ICA vs PCA	12
4.5 Módulo de Modelos (<code>models</code>)	12
4.5.1 Clasificadores Disponibles	13
4.5.2 Sistema de Modelos Personalizados	13
4.5.3 Persistencia de Modelos	14
4.6 Módulo de Entrenamiento (<code>training</code>)	14
4.6.1 Componentes	14
4.6.2 Pipeline de Experimentación Riguroso	15
4.7 Módulo de Evaluación (<code>evaluation</code>)	15
4.7.1 Métricas de Clasificación	15
4.7.2 Análisis Especializados	16
4.8 Módulo de Explicabilidad (<code>explainability</code>)	16
4.8.1 Componentes	16

4.8.2	Optimización Inversa	17
4.9	Módulo de AutoML (<code>automl</code>)	17
4.9.1	Backends Soportados	17
4.9.2	Estimadores FLAML	17
4.9.3	Presets Configurables	18
4.10	Módulo de Scores Clínicos (<code>scoring</code>)	18
4.10.1	GRACE Score	18
4.10.2	TIMI Score	19
5	Dashboard Streamlit	19
5.1	Arquitectura del Dashboard	19
5.2	Páginas del Dashboard	20
5.3	Características de la UI	20
6	Sistema de Testing	20
6.1	Arquitectura de Tests	20
6.2	Fixtures Principales	21
6.3	Cobertura de Tests	22
6.4	Ejecución de Tests	22
7	Despliegue y Containerización	22
7.1	Configuración Docker	22
7.1.1	Dockerfile Principal	22
7.1.2	Docker Compose	23
7.2	Comandos de Despliegue	24
8	Documentación	24
8.1	Sistema de Documentación	24
8.2	Documentos Disponibles	25
9	Configuración y Personalización	25
9.1	Configuración Global	25
9.2	Variables de Entorno	26
10	Rendimiento y Optimización	26
10.1	Estrategias de Optimización	26
10.2	Consideraciones de Escalabilidad	26
11	Limitaciones y Trabajo Futuro	27
11.1	Limitaciones Actuales	27
11.2	Roadmap Futuro	27

12 Conclusiones	27
A Glosario de Términos	28
B Referencias de API	29
B.1 Funciones Principales	29

1 Resumen Ejecutivo

Mortality AMI Predictor es una aplicación de Machine Learning de extremo a extremo diseñada para la predicción de mortalidad intrahospitalaria y arritmias ventriculares en pacientes con Infarto Agudo de Miocardio (IAM). El sistema integra las mejores prácticas de ingeniería de software con metodologías rigurosas de ciencia de datos para proporcionar una herramienta robusta, interpretable y clínicamente útil.

1.1 Características Principales

- **Limpieza automatizada de datos** con múltiples estrategias de imputación, detección de outliers y codificación de variables categóricas.
- **Análisis exploratorio interactivo** univariado, bivariado y multivariado con visualizaciones Plotly.
- **Entrenamiento de múltiples modelos** incluyendo KNN, Regresión Logística, Árboles de Decisión, XGBoost, LightGBM y Redes Neuronales.
- **Sistema AutoML** integrado con FLAML y soporte opcional para Auto-sklearn y AutoKeras (NAS).
- **Evaluación exhaustiva** con métricas, curvas de calibración, Decision Curve Analysis y comparación con scores clínicos.
- **Explicabilidad avanzada** mediante SHAP, importancia por permutación, PDP y optimización inversa.
- **Scores clínicos integrados** (GRACE, TIMI) para validación y comparación.
- **Sistema de modelos personalizados** que permite crear e integrar arquitecturas propias.
- **Dashboard interactivo** multipágina en Streamlit con generación de reportes PDF.
- **Despliegue containerizado** con Docker y Docker Compose.

1.2 Métricas de Calidad del Código Fuente

Métrica	Valor
Número de módulos	10
Archivos Python especializados	38+
Tests automatizados	120+
Cobertura de tests	> 80 %
Type hints	100 %
Documentación (archivos .md)	25+

Cuadro 1: Métricas de calidad del código fuente

2 Arquitectura del Sistema

2.1 Visión General de la Arquitectura

El sistema sigue una arquitectura modular organizada en capas con clara separación de responsabilidades:

1. **Capa de Presentación:** Dashboard Streamlit multipágina con 10 páginas especializadas.
2. **Capa de Lógica de Negocio:** Módulos Python especializados en `src/`.
3. **Capa de Datos:** Sistema de gestión de archivos con timestamps y versionado automático.
4. **Capa de Persistencia:** Almacenamiento de modelos, datasets y configuraciones.

2.2 Estructura de Directorios

```

1 Tools/
2   +- dashboard/           # Aplicacion Streamlit
3   |   +- Dashboard.py    # Punto de entrada principal
4   |   +- app/              # Utilidades y estado compartido
5   |   +- pages/            # Paginas del dashboard (10)
6   +- src/                 # Modulos de logica de negocio
7   |   +- automl/           # Integracion AutoML
8   |   +- cleaning/          # Limpieza de datos
9   |   +- data_load/         # Carga y gestion de datos
10  |   +- eda/               # Analisis exploratorio

```

```

11 |     +-+ evaluation/           # Evaluacion de modelos
12 |     +-+ explainability/    # Explicabilidad (SHAP, etc.)
13 |     +-+ features/          # Ingenieria de caracteristicas
14 |     +-+ models/             # Definicion de modelos
15 |     +-+ prediction/        # Sistema de predicción
16 |     +-+ preprocessing/     # Pipelines de preprocesamiento
17 |     +-+ reporting/         # Generacion de reportes PDF
18 |     +-+ scoring/           # Scores clinicos
19 |     +-+ training/          # Entrenamiento de modelos
20 |   +-+ tests/               # Suite de tests automatizados
21 |   +-+ docker/              # Configuracion Docker
22 |   +-+ docs/                # Documentacion MkDocs
23 |   +-+ processed/           # Datos procesados y modelos

```

Listing 1: Estructura del proyecto

2.3 Patrones de Diseño Implementados

El sistema implementa múltiples patrones de diseño profesionales:

Patrón	Aplicación
Factory	Creación de clasificadores y configuraciones de preprocesamiento
Strategy	Estrategias de imputación, encoding, detección de outliers
Builder	Construcción de pipelines de preprocesamiento
Singleton	Configuración global del proyecto (CONFIG)
Adapter	Integración de diferentes backends AutoML
Registry	Registro de modelos personalizados y scores clínicos
Template Method	Clase base para modelos custom (BaseCustomModel)

Cuadro 2: Patrones de diseño implementados en el sistema

3 Tecnologías y Dependencias

3.1 Stack Tecnológico Principal

Categoría	Tecnología	Propósito
Lenguaje	Python 3.9+ Type Hints	Lenguaje principal de desarrollo Tipado estático opcional
ML Core	Scikit-learn	Algoritmos de ML y pipelines
	XGBoost	Gradient boosting optimizado
	LightGBM	Gradient boosting eficiente
	Imbalanced-learn	Manejo de clases desbalanceadas
Deep Learning	PyTorch AutoKeras/TensorFlow	Redes neuronales tabulares Neural Architecture Search
AutoML	FLAML Auto-sklearn	AutoML cross-platform AutoML (solo Linux)
Visualización	Plotly Matplotlib Seaborn	Gráficos interactivos Gráficos estáticos Visualización estadística
Explicabilidad	SHAP Scikit-plot	Valores Shapley Curvas ROC, calibración
UI/Dashboard	Streamlit	Dashboard web interactivo
Reportes	ReportLab	Generación de PDFs
Tracking	MLflow	Seguimiento de experimentos
Testing	Pytest Pytest-cov	Framework de testing Cobertura de código
Despliegue	Docker Docker Compose	Containerización Orquestación de servicios

Cuadro 3: Stack tecnológico completo del sistema

3.2 Dependencias Principales

```

1 pandas                      # Manipulacion de datos
2 numpy                        # Computacion numerica
3 scikit-learn                 # Machine Learning
4 xgboost                       # Gradient Boosting
5 lightgbm                      # Light Gradient Boosting
6 imbalanced-learn              # Manejo de clases desbalanceadas
7 optuna                         # Optimizacion de hiperparametros
8 matplotlib                    # Visualizacion

```

```

9 seaborn          # Visualizacion estadistica
10 plotly           # Graficos interactivos
11 shap              # Explicabilidad
12 streamlit        # Dashboard web
13 joblib            # Serializacion de modelos
14 mlflow            # Tracking de experimentos
15 reportlab         # Generacion de PDFs
16 scipy              # Computacion cientifica
17 flaml[automl]     # AutoML cross-platform
18 torch              # Deep Learning (opcional)

```

Listing 2: Dependencias principales (requirements.txt)

4 Módulos Funcionales

4.1 Módulo de Carga de Datos (data_load)

El módulo `data_load` proporciona utilidades robustas para la carga, guardado y gestión de datasets.

4.1.1 Componentes

- **loaders.py:** Carga de datos con detección automática de formato (CSV, Parquet, Feather, Excel) y manejo robusto de codificaciones.
- **io_utils.py:** Utilidades de I/O con detección automática de encoding para archivos con caracteres especiales (español).
- **splitters.py:** Estrategias de división de datos:
 - División aleatoria estratificada
 - División temporal (para datos ordenados en el tiempo)
 - División con preservación de distribución de clases
- **path_utils.py:** Gestión de rutas con timestamps automáticos, limpieza de archivos antiguos y organización de outputs.

4.1.2 Características Técnicas

```

1 from src.data_load import load_dataset, train_test_split
2
3 # Carga automatica con detección de formato
4 df = load_dataset("path/to/data.csv") # Soporta CSV, Parquet, Excel
5
6 # División estratificada
7 train_df, test_df = train_test_split(
8     df,
9     test_size=0.2,
10    stratify_column="mortality"
11)
12
13 # División temporal
14 train_df, test_df = train_test_split(
15     df,
16     time_column="admission_date",
17     test_size=0.2
18)

```

Listing 3: Ejemplo de uso del loader de datos

4.2 Módulo de Limpieza de Datos (cleaning)

El módulo `cleaning` implementa un pipeline completo de limpieza de datos con tracking de metadatos.

4.2.1 Componentes

- **cleaner.py:** Clase orquestadora `DataCleaner` que coordina todas las operaciones de limpieza.
- **imputation.py:** Estrategias de imputación:
 - Media, mediana, moda
 - KNN Imputer
 - Forward/Backward fill
 - Valores constantes personalizables
- **outliers.py:** Métodos de detección y tratamiento de outliers:
 - IQR (Rango Intercuartílico)

- Z-score
 - Tratamientos: cap, remove, flag
- **encoding.py:** Codificación de variables categóricas:
- Label Encoding
 - One-Hot Encoding
 - Ordinal Encoding (con orden personalizado)
- **discretization.py:** Discretización de variables continuas:
- Equal-width binning
 - Equal-frequency binning
 - K-Means binning
 - Bins personalizados
- **metadata.py:** Tracking de metadatos de variables (tipo original, transformaciones aplicadas, estadísticas).

4.2.2 Configuración de Limpieza

```
1 from src.cleaning import DataCleaner, CleaningConfig
2
3 config = CleaningConfig(
4     # Imputacion
5     numeric_imputation="median",
6     categorical_imputation="mode",
7     knn_neighbors=5,
8
9     # Outliers
10    outlier_method="iqr",
11    iqr_multiplier=1.5,
12    outlier_treatment="cap",
13
14    # Encoding
15    categorical_encoding="label",
16
17    # General
18    drop_duplicates=True,
19    drop_fully_missing=True,
20    drop_constant=True
21)
22
23 cleaner = DataCleaner(config)
```

```
24 df_clean = cleaner.fit_transform(df, target_column="mortality")
```

Listing 4: Configuración del pipeline de limpieza

4.3 Módulo de Análisis Exploratorio (eda)

El módulo `eda` proporciona un análisis exploratorio completo con visualizaciones interactivas.

4.3.1 Componentes

- **analyzer.py:** Clase principal `EDAAalyzer` que orquesta el análisis.
- **univariate.py:** Análisis univariado (estadísticas descriptivas, distribuciones, detección de anomalías).
- **bivariate.py:** Análisis bivariado (correlaciones, tests estadísticos, scatter plots).
- **multivariate.py:** Análisis multivariado (PCA, matrices de correlación, análisis de componentes).
- **visualizations.py:** Generación de visualizaciones interactivas con Plotly.
- **pdf_reports.py:** Generación de reportes PDF profesionales de EDA.

4.3.2 Estadísticas Computadas

Tipo de Análisis	Métricas/Visualizaciones
Univariado Numérico	Media, mediana, desviación estándar, IQR, skewness, kurtosis, histogramas, boxplots
Univariado Categórico	Frecuencias, proporciones, moda, gráficos de barras, pie charts
Bivariado Num-Num	Correlación Pearson/Spearman, scatter plots, regresión lineal
Bivariado Num-Cat	ANOVA, t-test, violin plots, boxplots agrupados
Bivariado Cat-Cat	Chi-cuadrado, Cramér's V, tablas de contingencia
Multivariado	PCA, matrices de correlación, pair plots

Cuadro 4: Análisis estadísticos disponibles en el módulo EDA

4.4 Módulo de Características (features)

El módulo `features` implementa técnicas avanzadas de ingeniería y transformación de características.

4.4.1 Componentes

- **selectors.py:** Selección segura de columnas excluyendo identificadores y targets.
- **ica.py:** Transformación mediante Análisis de Componentes Independientes (ICA):
 - Separación de señales mixtas
 - Extracción de componentes no-Gaussianos
 - Comparación con PCA
 - Visualizaciones de componentes
- **transformers.py:** Transformadores personalizados compatibles con sklearn pipelines.

4.4.2 ICA vs PCA

El sistema permite comparar ambas técnicas de reducción de dimensionalidad:

```

1 from src.features import ICATransformer, compare_pca_vs_ica
2
3 # Transformacion ICA
4 ica = ICATransformer(n_components=10, algorithm='parallel')
5 ica.fit(X_train)
6 X_transformed = ica.transform(X_train)
7
8 # Comparacion PCA vs ICA
9 comparison_fig = compare_pca_vs_ica(X_train, n_components=10)

```

Listing 5: Uso de ICA para transformación de features

4.5 Módulo de Modelos (models)

El módulo `models` define los clasificadores disponibles y el sistema de modelos personalizados.

4.5.1 Clasificadores Disponibles

Modelo	Clave	Características
K-Nearest Neighbors	knn	Algoritmo basado en instancias
Regresión Logística	logreg	Modelo lineal calibrado
Árbol de Decisión	dtree	Modelo interpretable
XGBoost	xgb	Gradient boosting optimizado
XGBoost Balanced	xgb_balanced	Ponderación para clases desbalanceadas
LightGBM	lgbm	Gradient boosting eficiente
Red Neuronal	nn	MLP con PyTorch

Cuadro 5: Clasificadores estándar disponibles

4.5.2 Sistema de Modelos Personalizados

El sistema permite crear modelos personalizados que se integran completamente con el pipeline:

```

1 from src.models.custom_base import BaseCustomClassifier
2 import numpy as np
3
4 class MyCustomClassifier(BaseCustomClassifier):
5     def __init__(self, n_layers=3, learning_rate=0.01):
6         super().__init__(name="MyCustomClassifier")
7         self.n_layers = n_layers
8         self.learning_rate = learning_rate
9
10    def fit(self, X, y):
11        self._validate_input(X, training=True)
12        self.classes_ = np.unique(y)
13        # Logica de entrenamiento
14        self.is_fitted_ = True
15        return self
16
17    def predict(self, X):
18        self._validate_input(X)
19        # Logica de prediccion
20        return predictions
21
22    def predict_proba(self, X):
23        # Probabilidades
24        return probabilities

```

Listing 6: Creación de un modelo personalizado

4.5.3 Persistencia de Modelos

El sistema de persistencia incluye:

- Serialización con joblib y pickle
- Metadatos del modelo (hiperparámetros, métricas, fecha)
- Versionado con timestamps
- Validación de integridad (hash SHA256)
- Soporte para pipelines de preprocesamiento

4.6 Módulo de Entrenamiento (training)

El módulo **training** implementa un pipeline de entrenamiento riguroso siguiendo estándares académicos.

4.6.1 Componentes

- **trainer.py**: Orquestador principal del pipeline de experimentación.
- **cross_validation.py**: Estrategias de validación cruzada:
 - Nested Cross-Validation
 - Repeated Stratified K-Fold ($10 \times 10 = 100$ runs)
- **hyperparameter_tuning.py**: Búsqueda de hiperparámetros con RandomizedSearchCV y GridSearchCV.
- **statistical_tests.py**: Tests estadísticos para comparación de modelos (t-test, Mann-Whitney).
- **learning_curves.py**: Generación de curvas de aprendizaje.
- **pdf_reports.py**: Reportes PDF de entrenamiento.

4.6.2 Pipeline de Experimentación Riguroso

1. FASE 1 - Train + Validation:

- Validación cruzada estratificada repetida (≥ 30 runs)
- Estimación de μ (media) y σ (desviación estándar)
- Curvas de aprendizaje para cada modelo

2. FASE 2 - Test:

- Bootstrap resampling en conjunto de test
- Jackknife resampling (leave-one-out)
- Intervalos de confianza

3. FASE 3 - Comparación Estadística:

- Test de normalidad Shapiro-Wilk
- Paired t-test (si normal) o Mann-Whitney U (si no normal)
- Determinación de diferencias significativas

4.7 Módulo de Evaluación (evaluation)

El módulo `evaluation` proporciona métricas exhaustivas y análisis de rendimiento.

4.7.1 Métricas de Clasificación

Métrica	Descripción
AUROC	Área bajo la curva ROC
AUPRC	Área bajo la curva Precision-Recall
Accuracy	Exactitud general
Precision	Precisión (valor predictivo positivo)
Recall	Sensibilidad (tasa de verdaderos positivos)
Specificity	Especificidad (tasa de verdaderos negativos)
F1-Score	Media armónica de precision y recall
NPV	Valor predictivo negativo
Brier Score	Error cuadrático de calibración

Cuadro 6: Métricas de clasificación implementadas

4.7.2 Análisis Especializados

- **Calibración:** Curvas de calibración con múltiples estrategias de binning.
- **Decision Curve Analysis:** Evaluación de utilidad clínica vs. estrategias “treat all” y “treat none”.
- **Comparación con GRACE:** Benchmark contra el score clínico estándar.
- **Bootstrap Validation:** Intervalos de confianza robustos.

4.8 Módulo de Explicabilidad (explainability)

El módulo `explainability` implementa técnicas de interpretación de modelos de ML.

4.8.1 Componentes

- **shap_analysis.py:** Análisis SHAP completo:
 - TreeExplainer para modelos basados en árboles
 - General Explainer para cualquier modelo
 - Beeswarm plots, bar plots, waterfall plots, force plots
 - Importancia de features basada en SHAP
- **permutation.py:** Importancia por permutación.
- **partial_dependence.py:** Partial Dependence Plots (PDP) para visualizar efectos marginales.
- **inverse_optimization.py:** Optimización Inversa para recomendaciones de tratamiento:
 - Encuentra valores óptimos de features modificables
 - Minimiza/maximiza predicción del modelo
 - Respeta restricciones clínicas
 - Genera recomendaciones actionable
- **pdf_reports.py:** Reportes PDF de explicabilidad.

4.8.2 Optimización Inversa

Esta funcionalidad innovadora permite responder preguntas como: “¿Qué valores de presión arterial y medicación optimizarían el pronóstico de este paciente?”

```

1 from src.explainability import InverseOptimizer
2
3 optimizer = InverseOptimizer(
4     model=trained_model,
5     feature_names=feature_cols,
6     feature_bounds={'sbp': (80, 180), 'hr': (40, 150)}
7 )
8
9 result = optimizer.optimize(
10    target_value=0, # Mortalidad = 0
11    modifiable_features=['sbp', 'hr', 'medication_X'],
12    fixed_features={'age': 65, 'sex': 1}
13 )
14
15 print(f"Valores optimos: {result['optimal_values']}")
16 print(f"Reducción de riesgo: {result['risk_reduction']}")

```

Listing 7: Uso de optimización inversa

4.9 Módulo de AutoML (automl)

El módulo `automl` integra múltiples frameworks de AutoML.

4.9.1 Backends Soportados

Backend	Plataforma	Características
FLAML	Cross-platform	Rápido, ligero, 12+ estimadores
Auto-sklearn	Linux/WSL	Completo, meta-learning
AutoKeras	Cross-platform	Neural Architecture Search

Cuadro 7: Backends AutoML disponibles

4.9.2 Estimadores FLAML

- **Gradient Boosting:** LightGBM, XGBoost, CatBoost, HistGB

- **Ensemble:** Random Forest, Extra Trees
- **Lineales:** Logistic Regression (L1/L2), SGD, SVC
- **Instance-Based:** K-Nearest Neighbors
- **Neural:** Multi-Layer Perceptron

4.9.3 Presets Configurables

```

1 from src.automl import FLAMLClassifier, AutoMLPreset
2
3 # Preset rápido (5 minutos)
4 clf_quick = FLAMLClassifier(
5     time_budget=300,
6     metric='roc_auc',
7     preset=AutoMLPreset.QUICK
8 )
9
10 # Preset balanceado (1 hora)
11 clf_balanced = FLAMLClassifier(
12     time_budget=3600,
13     metric='roc_auc',
14     estimator_list=['lgbm', 'xgboost', 'rf', 'catboost']
15 )
16
17 clf_balanced.fit(X_train, y_train)

```

Listing 8: Configuración de AutoML

4.10 Módulo de Scores Clínicos (scoring)

El módulo **scoring** implementa calculadores de scores clínicos validados.

4.10.1 GRACE Score

El score GRACE (Global Registry of Acute Coronary Events) predice mortalidad intra-hospitalaria y a 6 meses:

- **Variables:** Edad, frecuencia cardíaca, presión arterial sistólica, creatinina, clase Killip, desviación ST, enzimas cardíacas elevadas, parada cardíaca.
- **Output:** Score numérico y categoría de riesgo (bajo/intermedio/alto).

4.10.2 TIMI Score

El score TIMI (Thrombolysis In Myocardial Infarction):

- **Variables:** Edad, diabetes, hipertensión, infarto previo, y otros factores de riesgo.
- **Output:** Score y categoría de riesgo.

5 Dashboard Streamlit

5.1 Arquitectura del Dashboard

El dashboard está construido como una aplicación multipágina de Streamlit con las siguientes características:

- **Página principal:** Resumen del sistema y estadísticas rápidas.
- **10 páginas especializadas:** Cada una con funcionalidad específica.
- **Gestión de estado:** Session state para persistencia de datos entre páginas.
- **Configuración centralizada:** Rutas, temas y opciones en `app/config.py`.

5.2 Páginas del Dashboard

#	Página	Funcionalidad
00	Data Cleaning & EDA	Limpieza de datos y análisis exploratorio completo
01	Data Overview	Resumen y estadísticas del dataset
02	Model Training	Entrenamiento de modelos con validación cruzada
03	Predictions	Predicciones individuales y por lotes
04	Model Evaluation	Métricas, ROC, calibración, DCA
05	Explainability	SHAP, permutación, PDP
06	Clinical Scores	Cálculo de GRACE y TIMI
07	Custom Models	Creación y gestión de modelos personalizados
08	Inverse Optimization	Optimización de features para mejorar pronóstico
09	AutoML	Entrenamiento automático con FLAML/AutoKeras

Cuadro 8: Páginas del dashboard Streamlit

5.3 Características de la UI

- **Visualizaciones interactivas:** Gráficos Plotly con zoom, pan y exportación.
- **Formularios dinámicos:** Configuración de parámetros con validación.
- **Progreso en tiempo real:** Barras de progreso durante entrenamiento.
- **Exportación de reportes:** Generación de PDFs profesionales.
- **Caching inteligente:** Uso de `@st.cache_data` para optimización.

6 Sistema de Testing

6.1 Arquitectura de Tests

El sistema utiliza pytest con una arquitectura de tests organizada:

```
1 tests/
```

```

2  +- conftest.py                         # Fixtures compartidos
3  +- test_data_cleaning.py               # Tests de limpieza (25 tests)
4  +- test_modular_structure.py          # Tests de estructura (13 tests)
5  +- test_data_preprocess.py            # Tests de preprocesamiento
6  +- test_model_io.py                  # Tests de I/O de modelos
7  +- test_smoke_train.py               # Smoke test de entrenamiento
8  +- test_custom_models.py             # Tests de modelos custom (30+ tests)
9  +- test_model_serialization.py       # Tests de serializacion (25+ tests)
10 +- test_integration_pipeline.py      # Tests end-to-end (20+ tests)
11 +- test_inverse_optimization.py      # Tests de optimizacion inversa
12 +- test_automl.py                   # Tests de AutoML
13 +- TESTING_SUMMARY.md                # Documentacion de tests

```

Listing 9: Estructura de tests

6.2 Fixtures Principales

```

1  @pytest.fixture
2  def clinical_like_dataset(random_seed):
3      """Dataset sintetico similar a datos clinicos."""
4      data = pd.DataFrame({
5          'age': np.random.randint(40, 90, 300),
6          'sbp': np.random.normal(120, 20, 300),
7          'heart_rate': np.random.randint(60, 120, 300),
8          'creatinine': np.random.normal(1.0, 0.5, 300),
9          'mortality': np.random.choice([0, 1], 300, p=[0.8, 0.2])
10     })
11     return data
12
13 @pytest.fixture
14 def trained_simple_model(simple_dataset):
15     """Modelo sklearn entrenado."""
16     X, y = simple_dataset
17     model = RandomForestClassifier(n_estimators=10)
18     model.fit(X.select_dtypes(include=[np.number]), y)
19     return model

```

Listing 10: Fixtures de pytest

6.3 Cobertura de Tests

Módulo	Tests	Estado
Data Cleaning	25	✓
Modular Structure	13	✓
Custom Models	30+	✓
Model Serialization	25+	✓
Integration Pipeline	20+	✓
AutoML	5+	✓
Inverse Optimization	10+	✓
Total	120+	✓

Cuadro 9: Distribución de tests por módulo

6.4 Ejecución de Tests

```

1 # Ejecutar todos los tests
2 pytest tests/ -v
3
4 # Con cobertura
5 pytest tests/ -v --cov=src --cov-report=html
6
7 # Solo tests rapidos
8 pytest tests/ -v -m "not slow"
9
10 # Tests de integracion
11 pytest tests/ -v -m integration

```

Listing 11: Comandos de testing

7 Despliegue y Containerización

7.1 Configuración Docker

El sistema incluye configuración completa de Docker para despliegue reproducible.

7.1.1 Dockerfile Principal

```

1 FROM python:3.11-slim
2

```

```

3 WORKDIR /app
4
5 # Dependencias del sistema
6 RUN apt-get update && apt-get install -y \
7     gcc g++ git curl libgomp1 libopenblas-dev swig
8
9 # Dependencias Python
10 COPY requirements.txt .
11 RUN pip install -r requirements.txt
12 RUN pip install "fmlaml[automl]"
13
14 # Código de la aplicación
15 COPY src/ ./src/
16 COPY dashboard/ ./dashboard/
17
18 # Directorios
19 RUN mkdir -p processed/models processed/plots models mlruns logs
20
21 EXPOSE 8501
22
23 CMD ["streamlit", "run", "dashboard/Dashboard.py", \
24       "--server.port=8501", "--server.address=0.0.0.0"]

```

Listing 12: Dockerfile simplificado

7.1.2 Docker Compose

```

1 version: '3.8'
2
3 services:
4     app:
5         build:
6             context: ..
7             dockerfile: docker/Dockerfile
8             container_name: mortality-ami-predictor
9         ports:
10            - "8501:8501"
11         volumes:
12            - ..../DATA:/app/DATA:ro
13            - ..../processed:/app/processed
14            - ..../models:/app/models
15         environment:
16            - AUTOML_BACKEND=fmlaml
17            - AUTOML_TIME_BUDGET=3600
18
19     jupyter:

```

```

20 # Servicio Jupyter para desarrollo
21 ports:
22   - "8888:8888"
23 profiles: [dev]
24
25 mlflow:
26   # Servicio MLflow para tracking
27   ports:
28     - "5000:5000"
29   profiles: [dev]
```

Listing 13: docker-compose.yml

7.2 Comandos de Despliegue

```

1 # Construir imagen
2 docker build -t mortality-ami-predictor .
3
4 # Ejecutar contenedor
5 docker run -p 8501:8501 mortality-ami-predictor
6
7 # Con Docker Compose (producción)
8 docker-compose up -d app
9
10 # Con Docker Compose (desarrollo)
11 docker-compose --profile dev up -d
```

Listing 14: Comandos de despliegue

8 Documentación

8.1 Sistema de Documentación

El proyecto utiliza MkDocs con Material theme para documentación técnica:

- **Guías de Usuario:** Tutoriales paso a paso para cada funcionalidad.
- **Referencia API:** Documentación automática de módulos Python.
- **Guías de Desarrollo:** Contribución, testing, arquitectura.
- **Changelog:** Historial de cambios por versión.

8.2 Documentos Disponibles

Documento	Contenido
index.md	Página principal con overview del proyecto
AUTOML_GUIDE.md	Guía completa de AutoML
CUSTOM_MODELS_GUIDE.md	Creación de modelos personalizados
CUSTOM_MODELS_ARCHITECTURE.md	Arquitectura del sistema de modelos
PDF_GENERATION_GUIDE.md	Generación de reportes PDF
ICA_UI_IMPLEMENTATION.md	Implementación de ICA en la UI
IMPLEMENTATION_SUMMARY.md	Resumen de implementación

Cuadro 10: Documentación principal del proyecto

9 Configuración y Personalización

9.1 Configuración Global

La configuración del proyecto se centraliza en `src/config.py`:

```

1 @dataclass
2 class ProjectConfig:
3     # Rutas de datos
4     dataset_path: str = os.environ.get("DATASET_PATH", "")
5     processed_dir: str = "processed"
6
7     # Columnas objetivo
8     target_column: str = "mortality_inhospital"
9     arrhythmia_column: str = "ventricular_arrhythmia"
10
11    # Tracking de experimentos
12    experiment_tracker: str = "mlflow"
13    tracking_uri: Optional[str] = None
14
15 CONFIG = ProjectConfig()
16 RANDOM_SEED = 42

```

Listing 15: Configuración global del proyecto

9.2 Variables de Entorno

Variable	Descripción
DATASET_PATH	Ruta al dataset principal
TARGET_COLUMN	Nombre de la columna objetivo
AUTOML_BACKEND	Backend AutoML (flaml, auto-sklearn)
AUTOML_TIME_BUDGET	Tiempo máximo para AutoML (segundos)
EXPERIMENT_TRACKER	Sistema de tracking (mlflow, wandb)

Cuadro 11: Variables de entorno configurables

10 Rendimiento y Optimización

10.1 Estrategias de Optimización

- **Caching:** Uso extensivo de `@st.cache_data` y `@st.cache_resource` en Streamlit.
- **Lazy Loading:** Carga diferida de módulos pesados (SHAP, PyTorch).
- **Parallel Processing:** Uso de `n_jobs=-1` en sklearn cuando es posible.
- **Optimización de memoria:** Uso de dtypes apropiados en pandas.
- **Batch Processing:** Predicciones y SHAP en lotes para grandes datasets.

10.2 Consideraciones de Escalabilidad

- Soporte para datasets de hasta 1M+ registros.
- Limpieza incremental para datasets muy grandes.
- Subsampling inteligente para análisis SHAP.
- Opciones de GPU para redes neuronales y AutoKeras.

11 Limitaciones y Trabajo Futuro

11.1 Limitaciones Actuales

1. **Auto-sklearn:** Solo disponible en Linux/WSL.
2. **GPU Support:** Limitado a PyTorch y TensorFlow/AutoKeras.
3. **Datos Temporales:** No hay soporte nativo para series temporales de ECG.
4. **Multi-idioma:** La UI está principalmente en inglés/español.

11.2 Roadmap Futuro

- Integración de modelos de deep learning para señales ECG.
- API REST para integración con sistemas hospitalarios.
- Soporte para datos longitudinales.
- Dashboard de monitoreo de modelos en producción.
- Integración con FHIR para interoperabilidad clínica.

12 Conclusiones

Mortality AMI Predictor representa una solución integral y profesional para la predicción de mortalidad intrahospitalaria en pacientes con infarto agudo de miocardio. El sistema destaca por:

1. **Arquitectura modular y extensible:** 10 módulos especializados con 38+ archivos y clara separación de responsabilidades.
2. **Rigor metodológico:** Pipeline de experimentación siguiendo estándares académicos con validación cruzada repetida, tests estadísticos y comparación con scores clínicos validados.
3. **Explicabilidad avanzada:** Integración completa de SHAP, importancia por permutación, PDP y optimización inversa para recomendaciones actionables.

4. **Flexibilidad:** Sistema de modelos personalizados, múltiples backends AutoML y soporte para diversos formatos de datos.
5. **Calidad de software:** 120+ tests automatizados, 100 % type hints, documentación completa y containerización para despliegue reproducible.
6. **Usabilidad:** Dashboard interactivo multipágina con visualizaciones Plotly y generación de reportes PDF profesionales.

El sistema está preparado para su uso en investigación clínica y, con las validaciones apropiadas, podría integrarse en entornos de práctica médica para apoyar la toma de decisiones clínicas en el manejo de pacientes con IAM.

A Glosario de Términos

AUROC Área Bajo la Curva ROC (Receiver Operating Characteristic)

AUPRC Área Bajo la Curva Precision-Recall

AutoML Machine Learning Automatizado

DCA Decision Curve Analysis

FLAML Fast and Lightweight AutoML

GRACE Global Registry of Acute Coronary Events

IAM Infarto Agudo de Miocardio

ICA Análisis de Componentes Independientes

NAS Neural Architecture Search

PDP Partial Dependence Plot

SHAP SHapley Additive exPlanations

SMOTE Synthetic Minority Over-sampling Technique

TIMI Thrombolysis In Myocardial Infarction

B Referencias de API

B.1 Funciones Principales

```
1 # Carga de datos
2 from src.data_load import load_dataset, train_test_split
3
4 # Limpieza
5 from src.cleaning import DataCleaner, CleaningConfig
6
7 # EDA
8 from src.eda import EDAAnalyzer, quick_eda
9
10 # Modelos
11 from src.models import make_classifiers, make_automl_classifiers
12
13 # Entrenamiento
14 from src.training import run_rigorous_experiment_pipeline
15
16 # Evaluacion
17 from src.evaluation import compute_classification_metrics
18
19 # Explicabilidad
20 from src.explainability import compute_shap_values, InverseOptimizer
21
22 # Scores clinicos
23 from src.scoring import GraceScore, TIMIScore
24
25 # AutoML
26 from src.automl import FLAMLClassifier, NASClassifier
```

Listing 16: API principal del sistema