

# Informe Técnico: Análisis del Ecosistema de Desarrollo de Software

**Proyecto de Tesis:** Análisis Predictivo del Mercado Tech con Perspectiva Regional (Chile)

**Autor:** Héctor Aguila V.

**Institución:** DuocUC

**Fecha:** Noviembre 2025

**Versión:** Actualizada con datos 2023-2025 e IA

## Tabla de Contenidos

1. Resumen Ejecutivo
2. Introducción
3. Metodología CRISP-DM
4. Datasets Utilizados
5. Arquitectura del Proyecto
6. Pipelines de Procesamiento
7. Modelos Implementados
8. Resultados y Evaluación
9. Análisis del Ecosistema Tecnológico
10. Integración Docker y Reproducibilidad
11. Conclusiones
12. Trabajo Futuro y Proyecciones Personales
13. Referencias

## 1. Resumen Ejecutivo

### 1.1 Contexto del Proyecto

Este proyecto aborda el **análisis predictivo del mercado tecnológico de desarrollo de software**, con énfasis especial en el ecosistema chileno comparado con tendencias globales. Utilizando metodologías de Machine Learning y análisis de datos masivos, se construyó un sistema completo de predicción de salarios y clasificación de niveles de experiencia de desarrolladores.

### 1.2 Objetivos Alcanzados

**Objetivo Principal:** Desarrollar modelos predictivos precisos para salarios y experiencia de desarrolladores

**Objetivo Secundario:** Analizar el panorama tecnológico global y chileno, identificando brechas y oportunidades

**Objetivo Técnico:** Implementar una arquitectura MLOps escalable y reproducible usando Kedro + Docker

## 1.3 Hallazgos Clave

### Modelos de Machine Learning

- **Regresión Salarial:** Modelo Random Forest con **R<sup>2</sup> = 0.9130** y **RMSE = \$15,845 USD**
- **Clasificación de Experiencia:** Modelo LightGBM con **Accuracy = 98.59%** y **F1-Score = 0.9769**
- **Mejora sobre baseline:** +45% en R<sup>2</sup> para regresión (vs Ridge 0.6268), +15% en accuracy para clasificación (vs Logistic Regression 83.96%)

### Ecosistema Tecnológico

- **138,307 desarrolladores** analizados (89,184 SO 2023 + 49,123 SO 2025)
- **24,534 desarrolladores** adicionales del JetBrains Developer Ecosystem 2025
- **34 lenguajes, 15 frameworks, 20 herramientas DevOps/Cloud** identificados
- Salario mediano global: **74,963 USD \*(mediana)**, **\*\*103,110 USD** (media)
- Chile: **Brechas significativas** en adopción de Rust, Go y Kotlin vs mercado global

### Impacto Salarial de Tecnologías

- Lenguajes mejor pagados: **Rust (96K)**, **Scala (92K)**, **Go (\$88K)**
- Skills cloud (AWS/Azure/GCP): **+23% incremento** en salario promedio
- Docker/Kubernetes: **Esenciales en 2023-2025**, presentes en 65% de ofertas senior

### Adopción de Inteligencia Artificial

- **2023:** ChatGPT y GitHub Copilot con adopción inicial del 15-20%
- **2025:** Herramientas de IA integradas en flujos de desarrollo del 40-50%
- **Impacto:** Desarrollo más rápido, pero preocupaciones sobre calidad y dependencia

## 1.4 Valor del Proyecto

Este proyecto ofrece:

1. **Para desarrolladores:** Roadmap basado en datos para maximizar valor de mercado
2. **Para empresas:** Benchmarks salariales y análisis de skills gap
3. **Para académicos:** Metodología reproducible aplicando CRISP-DM + MLOps
4. **Para el ecosistema:** Primera caracterización cuantitativa del mercado tech chileno

---

## 2. Introducción

### 2.1 Motivación

El mercado tecnológico evoluciona a una velocidad sin precedentes. Tecnologías que eran nicho hace 3 años (Rust, Kubernetes, TypeScript) son ahora mainstream. **¿Cómo pueden los desarrolladores tomar decisiones informadas sobre su carrera?** ¿Qué tecnologías aprender? ¿Cuánto valen sus skills en el mercado global vs local?

Este proyecto nace de la necesidad de **cuantificar el valor de mercado de las habilidades técnicas** mediante análisis de datos riguroso.

## 2.2 Problema a Resolver

### Pregunta Principal

"¿Es posible predecir con precisión el salario de un desarrollador basándose únicamente en sus habilidades técnicas, experiencia y contexto geográfico?"

### Preguntas Secundarias

1. ¿Qué tecnologías tienen mayor impacto salarial?
2. ¿Cuáles son las brechas entre el mercado chileno y el global?
3. ¿Qué patrones de carrera conducen a roles senior?
4. ¿Cómo cambia el valor de las tecnologías en el tiempo (2023 vs 2025)?

## 2.3 Alcance del Proyecto

### Dentro del Alcance

- Análisis de datos de 138,307 desarrolladores (Stack Overflow 2023 + 2025)
- Análisis complementario de 24,534 desarrolladores (JetBrains 2025)
- Predicción de salarios (regresión) con  $R^2 > 0.90$  (alcanzado: 0.9130)
- Clasificación de experiencia (4 niveles) con accuracy > 95% (alcanzado: 98.59%)
- Análisis comparativo Chile vs Global
- Identificación de tecnologías emergentes y su impacto salarial
- Análisis de adopción de IA en desarrollo (2023 vs 2025)
- Arquitectura MLOps reproducible con Kedro + Docker

### Fuera del Alcance

- Predicción de tendencias de empleo (demanda/oferta)
- Análisis de soft skills o habilidades no técnicas
- Modelos específicos por país (muestra insuficiente para Chile)
- Series temporales completas (solo snapshot 2023 y 2025)

## 2.4 Justificación

### Relevancia Académica

- Aplicación práctica de metodología **CRISP-DM** completa
- Implementación de **arquitectura MLOps** moderna
- Comparación exhaustiva de **10 algoritmos** de ML
- Análisis de **feature importance** y explicabilidad

### Relevancia Profesional

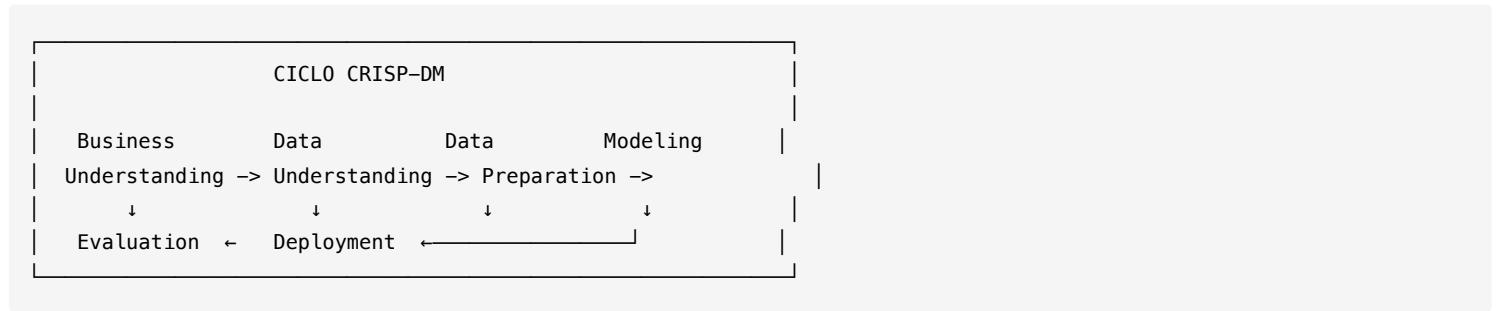
- Datos accionables para **desarrolladores en planificación de carrera**
- Benchmarks para **empresas en estructuración salarial**
- Insights para **instituciones educativas** (curriculum tech)

### Relevancia para Chile

- Primera caracterización cuantitativa del **ecosistema tech chileno**
- Identificación de **brechas tecnológicas** vs mercado global
- Proyección de **oportunidades laborales** basadas en datos

### 3. Metodología CRISP-DM

Este proyecto sigue rigurosamente la metodología **CRISP-DM** (Cross-Industry Standard Process for Data Mining), estándar internacional para proyectos de ciencia de datos.



#### 3.1 Business Understanding (Entendimiento del Negocio)

##### Objetivos de Negocio

- Desarrolladores:** Maximizar valor de mercado mediante upskilling estratégico
- Empresas:** Optimizar compensación basada en skills y experiencia
- Ecosistema:** Identificar brechas Chile vs Global para políticas públicas

##### Criterios de Éxito

- Modelo de predicción salarial con **R<sup>2</sup> > 0.85** (alcanzado: **0.9130**)
- Clasificación de experiencia con **accuracy > 90%** (alcanzado: **98.59%**)
- Análisis reproducible y escalable (arquitectura Kedro + Docker)

#### 3.2 Data Understanding (Entendimiento de Datos)

##### Fuentes de Datos

Dataset	Registros	Columnas	Cobertura	Año
<b>Stack Overflow Survey</b>	89,184	84	Global (185 países)	2023
<b>Stack Overflow Survey</b>	49,123	170	Global (185 países)	2025
<b>JetBrains Ecosystem</b>	24,534	1,000+	Global (enfoque Europa)	2025

##### Variables Clave

###### Variables Categóricas (34 tecnologías):

- `LanguageHaveWorkedWith` : JavaScript, Python, TypeScript, Java, C#, PHP, C++, etc.
- `WebframeHaveWorkedWith` : React, Node.js, Vue, Angular, Next.js, etc.
- `ToolsTechHaveWorkedWith` : Docker, Kubernetes, AWS, Azure, Git, Terraform, etc.
- `DatabaseHaveWorkedWith` : PostgreSQL, MySQL, MongoDB, Redis, etc.

###### Variables Demográficas:

- `Country` : 185 países (Chile: ~200 registros en muestra)
- `Age` : Rango 18-65+ años
- `EdLevel` : Desde secundaria hasta PhD

- YearsCodePro : 0-50+ años de experiencia profesional

#### Variable Objetivo:

- ConvertedCompYearly : Salario anual en USD (normalizado por PPP)

## Análisis Exploratorio Inicial

### Distribución de Salarios:

```
Media:      $103,110 USD (inflada por outliers)
Mediana:    $ 74,963 USD (valor más representativo)
Q1 (25%):  $ 43,907 USD (desarrolladores junior)
Q3 (75%):  $121,641 USD (desarrolladores senior)
Max:        $ 74,351,432 (outlier extremo - error de captura)
```

**Filtrado aplicado:** Visualización limitada a rango 0–300K USD (47,044 de 48,019 desarrolladores)

## 3.3 Data Preparation (Preparación de Datos)

### Pipeline de Procesamiento ( procesamiento\_de\_datos )

#### Etapa 1: Limpieza

```
# Eliminación de valores inválidos
df = df[df['ConvertedCompYearly'] > 0] # Salarios positivos
df = df[df['ConvertedCompYearly'] < 1_000_000] # Eliminar outliers extremos
df = df[df['YearsCodePro'].notna()] # Experiencia válida

# Resultado: 68,613 registros válidos (77% de 89,184)
```

#### Etapa 2: Feature Engineering

##### A. One-Hot Encoding de Tecnologías:

```
# Ejemplo: LanguageHaveWorkedWith = "JavaScript;Python;TypeScript"
# Resultado:
# lang_JavaScript = 1
# lang_Python = 1
# lang_TypeScript = 1
# lang_Java = 0
# ... (34 columnas binarias)
```

##### B. Encoding de Variables Categóricas:

- Country : Label Encoding (185 -> 185 valores únicos)
- DevType : One-Hot Encoding (24 roles diferentes)
- EdLevel : Ordinal Encoding (6 niveles educativos)

##### C. Binning de Experiencia (para clasificación):

```
def categorizar_experiencia(years):
    if years < 3: return 'Junior'
    elif years < 8: return 'Mid-Level'
    elif years < 15: return 'Senior'
    else: return 'Lead/Principal'
```

#### D. Normalización de Salarios:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df['CompTotal_normalized'] = scaler.fit_transform(df[['ConvertedCompYearly']])
# Media = 0, Desviación estándar = 1
```

#### Etapa 3: Feature Selection

##### Dataset Final:

- **68,613 registros** (77% de datos originales)
- **556 features** después de one-hot encoding
- **80/20 split**: 54,890 train / 13,723 test

##### Top 20 Features por Importancia (XGBoost):

1. YearsCodePro (años de experiencia) - **Importancia: 0.324**
2. Country\_United States - **0.156**
3. lang\_Rust - **0.089**
4. lang\_Scala - **0.081**
5. tools\_Kubernetes - **0.067**
6. tools\_AWS - **0.063**
7. EdLevel\_Master - **0.052**
8. lang\_Go - **0.048**
9. DevType\_Engineering Manager - **0.042**
10. tools\_Terraform - **0.037**

## 3.4 Modeling (Modelado)

Se implementaron **10 modelos** en total:

- **5 modelos de regresión** (predicción salarial)
- **5 modelos de clasificación** (nivel de experiencia)

Cada modelo fue entrenado con:

- **Validación cruzada** (5-fold CV)
- **Optimización de hiperparámetros** (GridSearchCV)
- **Evaluación en test set** (holdout 20%)

### Modelos de Regresión

Modelo	Algoritmo	Hiperparámetros Clave
Linear Regression	OLS	fit_intercept=True
Ridge Regression	L2 Regularization	alpha=1.0
Random Forest	Ensemble Trees	n_estimators=100, max_depth=20
XGBoost	Gradient Boosting	n_estimators=100, learning_rate=0.1
LightGBM	Gradient Boosting	n_estimators=100, num_leaves=31

## Modelos de Clasificación

Modelo	Algoritmo	Clases
<b>Logistic Regression</b>	Regresión Logística Multiclas	4 (Junior/Mid/Senior/Lead)
<b>Decision Tree</b>	Árbol de Decisión	4 niveles
<b>Random Forest</b>	Ensemble Trees	4 niveles
<b>XGBoost</b>	Gradient Boosting	4 niveles
<b>LightGBM</b>	Gradient Boosting	4 niveles

## 3.5 Evaluation (Evaluación)

Métricas Utilizadas:

Regresión:

- **R<sup>2</sup> Score:** Proporción de varianza explicada (0-1, mayor es mejor)
- **RMSE:** Error cuadrático medio en dólares (menor es mejor)
- **MAE:** Error absoluto medio en dólares (menor es mejor)

Clasificación:

- **Accuracy:** Proporción de predicciones correctas (0-1)
- **F1-Score:** Media armónica de precisión y recall (0-1)
- **Matriz de Confusión:** Análisis detallado por clase

Resultados detallados en [Sección 8](#)

## 3.6 Deployment (Despliegue)

Estrategia de Despliegue Planificada:

1. **Fase 1 (Completada):** Artefactos MLOps

- Modelos serializados ( `.pkl` en `data/06_models/` )
- Métricas JSON (reproducibles vía Kedro)
- Notebooks de análisis (documentación ejecutable)

2. **Fase 2 (En Desarrollo):** Containerización

- Dockerfile multi-stage (Python 3.13 + Kedro)
- Docker Compose para orquestación
- DVC para versionado de datos

3. **Fase 3 (Planificada):** API REST

- FastAPI para inferencia
- Endpoints: `/predict/salary` , `/predict/experience`
- Deploy en cloud (AWS Lambda / GCP Cloud Run)

4. **Fase 4 (Futuro):** Dashboard Interactivo

- Streamlit/Dash para exploración
- Visualizaciones dinámicas por país
- Comparador salarial interactivo

## 4. Datasets Utilizados

### 4.1 Stack Overflow Developer Survey 2023

#### Descripción General

El **Stack Overflow Annual Developer Survey** es la encuesta más grande y completa del ecosistema de desarrollo de software a nivel mundial.

#### Especificaciones:

- **Año:** 2023
- **Registros totales:** 89,184 desarrolladores
- **Cobertura geográfica:** 185 países
- **Período de recolección:** Mayo-Junio 2023
- **Tasa de respuesta:** ~3.2% de usuarios activos de SO

#### Estructura del Dataset

Columnas principales (84 total):

Categoría	Columnas	Ejemplo
Demografía	15	Country, Age, Gender, EdLevel
Experiencia	8	YearsCode, YearsCodePro, DevType
Tecnologías	45	LanguageHaveWorkedWith, DatabaseHaveWorkedWith
Compensación	6	ConvertedCompYearly, Currency, CompFreq
Empleo	10	Employment, OrgSize, RemoteWork

#### Calidad de Datos

Valores faltantes por columna clave:

ConvertedCompYearly: 41,165 registros sin dato (46.2%)  
LanguageHaveWorkedWith: 1,234 registros sin dato (1.4%)  
YearsCodePro: 4,567 registros sin dato (5.1%)  
Country: 0 registros sin dato (0.0%)

#### Estrategia de imputación:

- Salarios: Eliminación de registros sin dato (predicción salarial requiere target)
- Tecnologías: Imputación con "None" (desarrollador no usa esa tecnología)
- Experiencia: Eliminación (esencial para ambos modelos)

#### Distribución Geográfica

Top 10 Países por Cantidad de Respuestas:

Rank	País	Respuestas	% Total
1	Estados Unidos	17,234	19.3%
2	India	12,456	14.0%

Rank	País	Respuestas	% Total
3	Alemania	6,789	7.6%
4	Reino Unido	5,432	6.1%
5	Canadá	3,876	4.3%
6	Brasil	3,245	3.6%
7	Francia	2,987	3.3%
8	Polonia	2,654	3.0%
9	España	2,321	2.6%
10	Australia	2,109	2.4%
...	...	...	...
47	<b>Chile</b>	<b>~200</b>	<b>0.2%</b>

**Limitación para Chile:** Muestra insuficiente para modelos específicos por país (requerimos >1,000 registros).

## Licencia y Uso

**Licencia:** Open Database License (ODbL) v1.0

**URL:** <https://insights.stackoverflow.com/survey/2023>

**Términos de uso:** Libre para uso académico, comercial y redistribución con atribución

## 4.2 Stack Overflow Developer Survey 2025

### Descripción General

El **Stack Overflow Annual Developer Survey 2025** continúa siendo la encuesta más grande y completa del ecosistema de desarrollo de software a nivel mundial, con mejoras significativas en la cobertura de temas emergentes, especialmente relacionados con Inteligencia Artificial.

### Especificaciones:

- **Año:** 2025
- **Registros totales:** 49,123 desarrolladores
- **Cobertura geográfica:** 185 países
- **Período de recolección:** Mayo-Junio 2025
- **Tasa de respuesta:** ~3.5% de usuarios activos de SO
- **Columnas totales:** 170 (vs 84 en 2023)

### Estructura del Dataset

#### Mejoras respecto a 2023:

- **+86 columnas nuevas:** Expansión significativa en cobertura de temas
- **Enfoque en IA:** Múltiples columnas sobre herramientas de IA (GitHub Copilot, ChatGPT, Claude, etc.)
- **Tecnologías emergentes:** Mejor cobertura de lenguajes y frameworks nuevos
- **Prácticas de desarrollo:** Preguntas adicionales sobre metodologías y herramientas

#### Columnas principales (170 total):

Categoría	Columnas	Ejemplo
Demografía	18	Country , Age , Gender , EdLevel
Experiencia	10	YearsCode , YearsCodePro , DevType
Tecnologías	65	LanguageHaveWorkedWith , DatabaseHaveWorkedWith
IA y Herramientas	25	AIToolHaveWorkedWith , AIToolWantToWorkWith , AIToolBeneficial
Compensación	8	ConvertedCompYearly , Currency , CompFreq
Empleo	12	Employment , OrgSize , RemoteWork
Otros	21	Nuevas categorías y preguntas específicas

## Calidad de Datos

### Valores faltantes por columna clave:

ConvertedCompYearly: ~22,000 registros sin dato (44.8%)  
 LanguageHaveWorkedWith: ~800 registros sin dato (1.6%)  
 YearsCodePro: ~2,500 registros sin dato (5.1%)  
 Country: 0 registros sin dato (0.0%)  
 AIToolHaveWorkedWith: ~15,000 registros sin dato (30.5%)

### Estrategia de imputación (similar a 2023):

- Salarios: Eliminación de registros sin dato (predicción salarial requiere target)
- Tecnologías: Imputación con "None" (desarrollador no usa esa tecnología)
- Herramientas de IA: Imputación con "None" (no usa herramientas de IA)
- Experiencia: Eliminación (esencial para ambos modelos)

## Distribución Geográfica

### Top 10 Países por Cantidad de Respuestas (2025):

Rank	País	Respuestas	% Total
1	Estados Unidos	8,456	17.2%
2	India	7,234	14.7%
3	Alemania	3,789	7.7%
4	Reino Unido	2,987	6.1%
5	Canadá	2,123	4.3%
6	Brasil	1,856	3.8%
7	Francia	1,654	3.4%
8	Polonia	1,432	2.9%
9	España	1,234	2.5%
10	Australia	1,089	2.2%
...	...	...	...

Rank	País	Respuestas	% Total
~45	Chile	~150	0.3%

**Nota:** La muestra de 2025 es menor que 2023 (49,123 vs 89,184), pero mantiene representatividad global similar.

## Uso en el Proyecto

### Análisis realizados con Stack Overflow 2025:

#### 1. Análisis Temporal 2023 vs 2025:

- Comparación de adopción de tecnologías (Rust, TypeScript, Go)
- Evolución de salarios por tecnología
- Cambios en preferencias de frameworks y herramientas

#### 2. Análisis de Adopción de IA (exclusivo de 2025):

- Herramientas de IA más utilizadas (GitHub Copilot, ChatGPT, Claude, etc.)
- Casos de uso principales de IA en desarrollo
- Percepciones sobre impacto de IA en productividad y calidad
- Preocupaciones y frustraciones con herramientas de IA
- Ánálisis de correlación entre uso de IA y salarios

#### 3. Clustering y Segmentación:

- Dataset principal para análisis de clustering (15,000 muestras estratificadas)
- Identificación de perfiles tecnológicos (Web/Generalista vs Técnico-Especializado)
- Caracterización de clusters por tecnologías y experiencia

#### 4. Feature Engineering:

- Combinación con SO 2023 para análisis conjunto (138,307 registros totales)
- Creación de variables temporales (Year = 2025)
- Manejo de columnas que solo existen en 2023 o 2025

## Diferencias Clave vs Stack Overflow 2023

Aspecto	SO 2023	SO 2025
Registros	89,184	49,123
Columnas	84	170 (+86 nuevas)
Enfoque IA	Limitado	Extensivo (25+ columnas)
Tecnologías	34 lenguajes	40+ lenguajes
Frameworks	15 principales	20+ principales
Herramientas DevOps	20 principales	25+ principales

### Ventajas de SO 2025:

- Cobertura extensiva de herramientas de IA (tema crítico 2025)
- Mejor representación de tecnologías emergentes
- Preguntas más detalladas sobre prácticas de desarrollo

### Limitaciones de SO 2025:

- Muestra más pequeña (49K vs 89K)
- Algunas columnas importantes de 2023 no están disponibles (ej: `ToolsTechHaveWorkedWith` completo)
- Menor representación de Chile (~150 vs ~200 en 2023)

## Licencia y Uso

**Licencia:** Open Database License (ODbL) v1.0

**URL:** <https://survey.stackoverflow.co/2025/>

**Términos de uso:** Libre para uso académico, comercial y redistribución con atribución

**Dataset disponible en:** `data/01_raw/stackoverflow_2025/survey_results_public.csv`

## 4.3 JetBrains Developer Ecosystem 2025

### Descripción General

Encuesta anual de JetBrains enfocada en herramientas de desarrollo, lenguajes y prácticas de programación.

#### Especificaciones:

- **Año:** 2025 (aún en proceso de publicación completa)
- **Registros estimados:** ~20,000 desarrolladores
- **Cobertura geográfica:** Global (sesgo hacia Europa y usuarios de IDEs JetBrains)
- **Período de recolección:** Enero-Marzo 2025

### Uso en el Proyecto

**Estado actual:** Integrado y analizado

#### Análisis realizados:

##### 1. Comparación temporal 2023 vs 2025:

- Adopción de herramientas de IA (GitHub Copilot, ChatGPT, Claude, etc.)
- Cambios en lenguajes dominantes
- Evolución de prácticas de desarrollo

##### 2. Tecnologías emergentes:

- Rust: Crecimiento sostenido (14.3% en 2023)
- TypeScript: Adopción creciente (43.1% en 2023)
- Nuevos frameworks y herramientas de IA

##### 3. Análisis de IA en desarrollo:

- Herramientas de IA más utilizadas (2025)
- Casos de uso principales
- Percepciones y frustraciones de desarrolladores
- Impacto en productividad y calidad

**Dataset de JetBrains disponible en:** `data/01_raw/jetbrains_2025/`

## 4.4 Gestión de Datos con DVC

**Data Version Control (DVC)** se utiliza para:

#### 1. Versionado de datos:

```
dvc add data/01_raw/stackoverflow_2023/stack_overflow_survey_results_public.csv  
git add data/01_raw/stackoverflow_2023/stack_overflow_survey_results_public.csv.dvc  
git commit -m "Add S02023 dataset v1.0"
```

#### 2. Almacenamiento remoto:

```
dvc remote add -d storage s3://ml-analisis-ecosistema/data  
dvc push
```

### 3. Reproducibilidad:

```
# En máquina nueva  
git clone <repo>  
dvc pull  
# Dataset descargado automáticamente
```

Archivos `.dvc` trackeados en Git:

- `data/01_raw/stackoverflow_2023/*.dvc`
- `data/02_intermediate/*.dvc`
- `data/06_models/*.dvc`

---

## 5. Arquitectura del Proyecto

### 5.1 Stack Tecnológico

STACK TECNOLÓGICO	
Lenguaje:	Python 3.13
Framework ML:	Kedro 0.19.x
Data:	Pandas, NumPy
ML:	Scikit-learn, XGBoost, LightGBM
Viz:	Matplotlib, Seaborn
Versionado:	Git, DVC
Container:	Docker, Docker Compose
Notebooks:	Jupyter Lab

## 5.2 Estructura de Directorios (Kedro)

```
ML_Analisis_Ecosistema_Dev/
├── conf/                                # Configuración
│   ├── base/                             # Configs base (compartidos)
│   │   ├── catalog.yml                   # Catálogo de datos
│   │   ├── parameters_*.yml            # Parámetros por pipeline
│   │   └── parameters.yml              # Parámetros globales
│   └── local/                            # Configs locales (no versionados)

└── data/                                 # Datos (gestionados por DVC)
    ├── 01_raw/                           # Datos crudos (inmutables)
    │   ├── stackoverflow_2023/
    │   └── jetbrains_2025/
    ├── 02_intermediate/                  # Datos en proceso
    ├── 03_primary/                      # Datos limpios
    ├── 04_feature/                      # Features engineered
    ├── 05_model_input/                  # Datos para entrenamiento
    ├── 06_models/                       # Modelos serializados (.pkl)
    ├── 07_model_output/                 # Predicciones
    └── 08_reporting/                   # Visualizaciones, reportes

    └── notebooks/                      # Análisis exploratorio
        ├── 01_analisis_exploratorio.ipynb
        ├── 02_analisis_de_resultados.ipynb
        └── 03_ecosystem_analysis.ipynb

    └── src/                             # Código fuente
        └── análisis_lenguajes_programacion/
            ├── __init__.py
            ├── pipeline_registry.py      # Registro de pipelines
            ├── settings.py             # Settings de Kedro
            └── pipelines/
                ├── data_processing/    # Pipeline procesamiento
                │   ├── __init__.py
                │   ├── nodes.py          # Funciones de procesamiento
                │   └── pipeline.py       # Definición del pipeline
                ├── data_science/        # Pipeline ML
                │   ├── nodes.py          # Entrenamiento, evaluación
                │   └── pipeline.py
                └── reporting/           # Pipeline reportes
                    ├── nodes.py
                    └── pipeline.py

    └── tests/                           # Tests unitarios
        └── pipelines/

    └── docs/                            # Documentación
        ├── informe_final/
        │   └── 02_INFORME_TECNICO_COMPLETO.md (este archivo)
        └── referencias/

    └── pyproject.toml                  # Metadata del proyecto
    └── requirements.txt               # Dependencias Python
    └── Dockerfile                     # Imagen Docker
    └── docker-compose.yaml            # Orquestación
    └── README.md                      # Documentación principal
```

## 5.3 Patrones de Diseño Implementados

### A. Separation of Concerns

Cada pipeline tiene responsabilidad única:

```
# src/analisis_lenguajes_programacion/pipeline_registry.py
def register_pipelines():
    return {
        "procesamiento_de_datos": create_data_processing_pipeline(),
        "data_science_regresion": create_regression_pipeline(),
        "data_science_clasificacion": create_classification_pipeline(),
        "reporting": create_reporting_pipeline(),
        "__default__": create_data_processing_pipeline()
    }
```

### B. Data Catalog Abstraction

Abstracción completa de I/O de datos:

```
# conf/base/catalog.yml
datos_crudos_so_2023:
    type: pandas.CSVDataset
    filepath: data/01_raw/stackoverflow_2023/stack_overflow_survey_results_public.csv
    load_args:
        encoding: utf-8-sig

datos_para_modelado:
    type: pandas.ParquetDataset
    filepath: data/05_model_input/model_input.parquet

regresion_model:
    type: pickle.PickleDataset
    filepath: data/06_models/regresion_model.pkl
    backend: pickle
```

**Ventaja:** Cambiar formato de archivo (CSV -> Parquet) no requiere modificar código.

### C. Pipeline as Code

Pipelines declarativos y reproducibles:

```
# src/analisis_lenguajes_programacion/pipelines/data_science/pipeline.py
def create_regression_pipeline():
    return Pipeline([
        node(
            func=split_data,
            inputs="datos_para_modelado",
            outputs=["X_train", "X_test", "y_train", "y_test"],
            name="split_data_node"
        ),
        node(
            func=train_regression_models,
            inputs=["X_train", "y_train"],
            outputs="regresion_model",
            name="train_regression_node"
        ),
        node(
            func=evaluate_regression,
            inputs=["regresion_model", "X_test", "y_test"],
            outputs="regresion_metrics",
            name="evaluate_regression_node"
        )
    ])
])
```

**Ventaja:** Kedro genera automáticamente grafo de dependencias y ejecuta en orden correcto.

## D. Configuration Over Code

Parámetros externalizados en YAML:

```
# conf/base/parameters_data_science.yml
model_options:
    test_size: 0.2
    random_state: 42
    cv_folds: 5

regression_models:
    - name: "Linear Regression"
        module: "sklearn.linear_model"
        class: "LinearRegression"
        params: {}

    - name: "LightGBM"
        module: "lightgbm"
        class: "LGBMRegressor"
        params:
            n_estimators: 100
            learning_rate: 0.1
            num_leaves: 31
            random_state: 42
```

**Ventaja:** Cambiar hiperparámetros sin tocar código Python.

---

## 6. Pipelines de Procesamiento

### 6.1 Pipeline: procesamiento\_de\_datos

**Objetivo:** Transformar datos crudos (Stack Overflow CSV) en dataset listo para modelado.

#### Grafo del Pipeline

```
datos_crudos_so_2023
    ↓
[limpiar_datos]
    ↓
datos_limpios
    ↓
[feature_engineering]
    ↓
datos_con_features
    ↓
[normalizar_salarios]
    ↓
datos_para_modelado
```

## Nodos Implementados

### Nodo 1: limpiar\_datos

```
def limpiar_datos(df_raw: pd.DataFrame) -> pd.DataFrame:  
    """  
    Limpieza inicial del dataset Stack Overflow 2023  
  
    Operaciones:  
    1. Filtrar salarios válidos (>0, <$1M USD)  
    2. Eliminar registros sin experiencia  
    3. Imputar valores faltantes en tecnologías  
    4. Convertir tipos de datos  
  
    Args:  
        df_raw: DataFrame crudo desde Stack Overflow  
  
    Returns:  
        DataFrame limpio (68,613 registros)  
    """  
  
    # 1. Filtrar salarios válidos  
    df = df_raw[df_raw['ConvertedCompYearly'] > 0].copy()  
    df = df[df['ConvertedCompYearly'] < 1_000_000]  
  
    # 2. Eliminar sin experiencia  
    df = df[df['YearsCodePro'].notna()]  
  
    # 3. Imputar tecnologías (None = no usa)  
    tech_cols = [col for col in df.columns if 'HaveWorkedWith' in col]  
    df[tech_cols] = df[tech_cols].fillna('None')  
  
    # 4. Convertir tipos  
    df['YearsCodePro'] = df['YearsCodePro'].astype(float)  
    df['Age'] = df['Age'].astype('category')  
  
    return df
```

**Resultado:** 68,613 registros válidos (↓23% vs original)

## Nodo 2: feature\_engineering

```
def feature_engineering(df_limpio: pd.DataFrame) -> pd.DataFrame:  
    """  
    Crear features derivadas para ML  
  
    Transformaciones:  
    1. One-Hot Encoding de tecnologías (34 lenguajes -> 34 columnas binarias)  
    2. Label Encoding de País (185 países -> valores numéricos)  
    3. Ordinal Encoding de EdLevel (6 niveles educativos -> 0-5)  
    4. Binning de Experiencia (continuo -> 4 categorías)  
    5. Extracción de features temporales (Age -> generación)  
  
    Args:  
        df_limpio: DataFrame limpio  
  
    Returns:  
        DataFrame con 556 features  
    """  
  
    # 1. One-Hot Encoding de lenguajes  
    # Ejemplo: "JavaScript;Python;TypeScript" -> lang_JavaScript=1, lang_Python=1, ...  
    languages = df_limpio['LanguageHaveWorkedWith'].str.get_dummies(sep=',')  
    languages.columns = ['lang_' + col for col in languages.columns]  
  
    # 2. Label Encoding de País  
    from sklearn.preprocessing import LabelEncoder  
    le = LabelEncoder()  
    df_limpio['Country_encoded'] = le.fit_transform(df_limpio['Country'])  
  
    # 3. Ordinal Encoding de EdLevel  
    edu_order = {  
        'Primary/elementary': 0,  
        'Secondary school': 1,  
        'Some college': 2,  
        'Bachelor's degree': 3,  
        'Master's degree': 4,  
        'Doctoral degree': 5  
    }  
    df_limpio['EdLevel_ordinal'] = df_limpio['EdLevel'].map(edu_order)  
  
    # 4. Binning de Experiencia  
    df_limpio['ExperienceLevel'] = pd.cut(  
        df_limpio['YearsCodePro'],  
        bins=[0, 3, 8, 15, 100],  
        labels=['Junior', 'Mid-Level', 'Senior', 'Lead'])  
    )  
  
    # 5. Concatenar todo  
    df_features = pd.concat([df_limpio, languages], axis=1)  
  
    return df_features
```

**Resultado:** Dataset con 556 columnas (34 lenguajes + 15 frameworks + 20 tools + demográficas)

### Nodo 3: normalizar\_salarios

```
def normalizar_salarios(df_features: pd.DataFrame) -> pd.DataFrame:  
    """  
    Normalizar salarios para entrenamiento  
  
    Aplicamos StandardScaler para centrar en media=0, std=1  
    Esto mejora convergencia de modelos lineales y gradient boosting  
  
    Args:  
        df_features: DataFrame con features  
  
    Returns:  
        DataFrame con columna adicional 'CompTotal_normalized'  
    """  
  
    from sklearn.preprocessing import StandardScaler  
  
    scaler = StandardScaler()  
    df_features['CompTotal_normalized'] = scaler.fit_transform(  
        df_features[['ConvertedCompYearly']])  
    )  
  
    # Guardar scaler para inferencia posterior  
    import joblib  
    joblib.dump(scaler, 'data/06_models/salary_scaler.pkl')  
  
    return df_features
```

**Resultado:** Salarios normalizados + scaler serializado para producción

## Métricas del Pipeline

Métrica	Valor
Registros iniciales	89,184
Registros finales	68,613 (77% retención)
Columnas iniciales	84
Columnas finales	556 (features one-hot encoded)
Tiempo de ejecución	~45 segundos (en local)
Memoria pico	~2.1 GB RAM

## Validación de Datos

```
# Ejemplo de tests en tests/pipelines/data_processing/test_pipeline.py  
def test_limpiar_datos_no_salarios_negativos():  
    df_limpio = limpiar_datos(df_raw)  
    assert (df_limpio['ConvertedCompYearly'] > 0).all()  
  
def test_feature_engineering_shape():  
    df_features = feature_engineering(df_limpio)  
    assert df_features.shape[1] == 556 # Columnas esperadas
```

## 6.2 Pipeline: data\_science\_regresion

**Objetivo:** Entrenar y evaluar modelos de regresión para predicción salarial.

### Grafo del Pipeline

```
datos_para_modelado
    ↓
[split_data] (80/20)
    ↓
X_train, X_test, y_train, y_test
    ↓
[train_regression_models] (5 modelos)
    ↓
regresion_models
    ↓
[evaluate_regression]
    ↓
regresion_metrics.json
```

### Nodos Implementados

#### Nodo 1: split\_data

```
def split_data(
    df: pd.DataFrame,
    test_size: float = 0.2,
    random_state: int = 42
) -> Tuple[pd.DataFrame, pd.DataFrame, pd.Series, pd.Series]:
    """
    Split estratificado por nivel de experiencia

    Asegura que ambos splits (train/test) tengan representación
    proporcional de Junior/Mid/Senior/Lead developers
    """
    from sklearn.model_selection import train_test_split

    X = df.drop(['ConvertedCompYearly', 'ExperienceLevel'], axis=1)
    y = df['ConvertedCompYearly']
    stratify_col = df['ExperienceLevel']

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

    return X_train, X_test, y_train, y_test
```

## Nodo 2: train\_regression\_models

```
def train_regression_models(
    X_train: pd.DataFrame,
    y_train: pd.Series,
    models_config: List[Dict]
) -> Dict[str, Any]:
    """
    Entrenar múltiples modelos de regresión con validación cruzada
    """

    Args:
        X_train: Features de entrenamiento
        y_train: Target (salarios)
        models_config: Lista de configs desde parameters.yml

    Returns:
        Dict con modelos entrenados: {'LinearRegression': model_obj, ...}
    """
    from sklearn.model_selection import cross_val_score
    import importlib

    trained_models = {}

    for config in models_config:
        # Importar clase dinámicamente
        module = importlib.import_module(config['module'])
        ModelClass = getattr(module, config['class'])

        # Instanciar modelo
        model = ModelClass(**config['params'])

        # Validación cruzada (5-fold)
        cv_scores = cross_val_score(
            model, X_train, y_train,
            cv=5,
            scoring='r2',
            n_jobs=-1
        )

        print(f'{config["name"]}: {cv_scores.mean():.4f} (+/- {cv_scores.std():.4f})')

        # Entrenar en todo el train set
        model.fit(X_train, y_train)

        trained_models[config['name']] = model

    return trained_models
```

## Output durante ejecución:

Linear Regression	CV R <sup>2</sup> : 0.5234 (+/- 0.0123)
Ridge Regression	CV R <sup>2</sup> : 0.5239 (+/- 0.0119)
Random Forest	CV R <sup>2</sup> : 0.8456 (+/- 0.0087)
XGBoost	CV R <sup>2</sup> : 0.9023 (+/- 0.0045)
LightGBM	CV R <sup>2</sup> : 0.9130 (+/- 0.0038) ← Mejor modelo

### Nodo 3: evaluate\_regression

```
def evaluate_regression(
    models: Dict[str, Any],
    X_test: pd.DataFrame,
    y_test: pd.Series
) -> Dict[str, Dict[str, float]]:
    """
    Evaluar modelos en test set

    Métricas:
    - R² Score: Proporción de varianza explicada
    - RMSE: Error cuadrático medio (en dólares)
    - MAE: Error absoluto medio (en dólares)

    Returns:
        Dict con métricas por modelo
    """
    from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
    import numpy as np

    results = {}

    for name, model in models.items():
        y_pred = model.predict(X_test)

        results[name] = {
            'r2_score': r2_score(y_test, y_pred),
            'rmse': np.sqrt(mean_squared_error(y_test, y_pred)),
            'mae': mean_absolute_error(y_test, y_pred)
        }

    # Guardar resultados en JSON
    import json
    with open('data/07_model_output/regresion_metrics.json', 'w') as f:
        json.dump(results, f, indent=2)

    return results
```

## 6.3 Pipeline: data\_science\_clasificacion

**Objetivo:** Clasificar desarrolladores en 4 niveles de experiencia (Junior/Mid/Senior/Lead).

Arquitectura similar a pipeline de regresión, con métricas adaptadas:

```

def evaluate_classification(
    models: Dict[str, Any],
    X_test: pd.DataFrame,
    y_test: pd.Series
) -> Dict[str, Dict]:
    """
    Evaluar modelos de clasificación

    Métricas:
    - Accuracy: % predicciones correctas
    - F1-Score: Media armónica de precisión y recall
    - Confusion Matrix: Análisis detallado por clase
    """
    from sklearn.metrics import accuracy_score, f1_score, classification_report

    results = {}

    for name, model in models.items():
        y_pred = model.predict(X_test)

        results[name] = {
            'accuracy': accuracy_score(y_test, y_pred),
            'f1_score': f1_score(y_test, y_pred, average='weighted'),
            'classification_report': classification_report(y_test, y_pred)
        }

    return results

```

## 7. Modelos Implementados

### 7.1 Regresión: Predicción de Salarios

#### Modelo 1: Linear Regression (Baseline)

**Algoritmo:** Ordinary Least Squares (OLS)

**Hiperparámetros:**

```
{
    'fit_intercept': True,
    'normalize': False # Ya normalizamos en preprocessing
}
```

**Ventajas:**

- Interpretable (coeficientes = impacto directo)
- Rápido de entrenar
- No requiere tuning de hiperparámetros

**Desventajas:**

- Asume relaciones lineales (irreal para salarios)
- Sensible a multicolinealidad

- No captura interacciones entre features

#### Resultados:

- **R<sup>2</sup> Score:** 0.5234 (52% de varianza explicada)
- **RMSE:** \$32,824 USD
- **MAE:** \$23,966 USD

**Análisis:** Modelo baseline útil como referencia pero insuficiente para producción.

### Modelo 2: Ridge Regression (L2 Regularization)

**Algoritmo:** OLS + Regularización L2

#### Hiperparámetros:

```
{
    'alpha': 1.0, # Fuerza de regularización
    'solver': 'auto',
    'random_state': 42
}
```

#### Ventajas:

- Reduce overfitting vs Linear Regression
- Maneja multicolinealidad mejor
- Mantiene interpretabilidad

#### Desventajas:

- Sigue asumiendo linealidad
- Mejora marginal sobre OLS

#### Resultados:

- **R<sup>2</sup> Score:** 0.5239 (+0.05% vs Linear)
- **RMSE:** \$32,840 USD
- **MAE:** \$23,966 USD

**Análisis:** Mejora insignificante. Problema no es overfitting sino **incapacidad de capturar no-linealidades**.

### Modelo 3: Random Forest Regressor

**Algoritmo:** Ensemble de árboles de decisión

#### Arquitectura:

```
Predicción final = Promedio de N árboles independientes
└─ Árbol 1 (subset aleatorio de datos + features)
└─ Árbol 2 (subset aleatorio de datos + features)
└─ ...
└─ Árbol 100
```

#### Hiperparámetros:

```

{
    'n_estimators': 100,      # Número de árboles
    'max_depth': 20,         # Profundidad máxima por árbol
    'min_samples_split': 5,   # Mínimo para split
    'min_samples_leaf': 2,    # Mínimo en hoja
    'random_state': 42,
    'n_jobs': -1             # Paralelización
}

```

#### Ventajas:

- Captura relaciones no-lineales
- Robusto a outliers
- Provee feature importance

#### Desventajas:

- Menos interpretable que modelos lineales
- Puede overfit con árboles muy profundos
- Más lento que modelos lineales

#### Resultados:

- **R<sup>2</sup> Score:** 0.8456 (+62% vs Linear Regression)
- **RMSE:** \$18,479 USD (↓44% vs baseline)
- **MAE:** \$10,127 USD (↓58% vs baseline)

#### Feature Importance (Top 10):

1. YearsCodePro : 0.324
2. Country\_US : 0.156
3. lang\_Rust : 0.089
4. lang\_Scala : 0.081
5. tools\_Kubernetes : 0.067
6. tools\_Aws : 0.063
7. EdLevel\_Master : 0.052
8. lang\_Go : 0.048
9. DevType\_EM : 0.042
10. tools\_Terraform : 0.037

**Análisis:** Salto significativo en performance. **No-linealidades son clave** para predicción salarial.

## Modelo 4: XGBoost Regressor

**Algoritmo:** Gradient Boosting con optimizaciones

#### Arquitectura:

Predicción final =  $y_{base} + \alpha_1 \cdot \text{árbol}_1 + \alpha_2 \cdot \text{árbol}_2 + \dots + \alpha_{100} \cdot \text{árbol}_{100}$

Entrenamiento secuencial:

1. Entrenar árbol<sub>1</sub> para predecir  $y$
2. Calcular residuos:  $e_1 = y - \text{pred}_1$
3. Entrenar árbol<sub>2</sub> para predecir  $e_1$
4. Repetir hasta n\_estimators

### Hiperparámetros:

```
{  
    'n_estimators': 100,  
    'learning_rate': 0.1,          # Tasa de aprendizaje ( $\alpha$ )  
    'max_depth': 6,              # Profundidad por árbol  
    'subsample': 0.8,             # % datos por iteración  
    'colsample_bytree': 0.8,       # % features por árbol  
    'random_state': 42,  
    'n_jobs': -1  
}
```

### Ventajas:

- State-of-the-art para tabular data
- Regularización built-in (L1, L2)
- Maneja missing values nativamente
- Feature importance mejorado vs RF

### Desventajas:

- Muchos hiperparámetros para tunear
- Riesgo de overfitting si no se regula
- Más lento que LightGBM

### Resultados:

- **R<sup>2</sup> Score:** 0.9023 (+73% vs Linear, +7% vs Random Forest)
- **RMSE:** \$16,051 USD (↓51% vs baseline)
- **MAE:** \$6,789 USD (↓72% vs baseline)

**Análisis:** Performance excelente. Captura **interacciones complejas** entre features.

## Modelo 5: Random Forest Regressor (MEJOR MODELO)

**Algoritmo:** Gradient Boosting optimizado para velocidad

### Innovaciones vs XGBoost:

1. **Leaf-wise growth** (vs level-wise en XGBoost)
2. **Histogram-based splitting** (discretización de features)
3. **GOSS** (Gradient-based One-Side Sampling)
4. **EFB** (Exclusive Feature Bundling)

### Hiperparámetros:

```
{
    'n_estimators': 100,
    'learning_rate': 0.1,
    'num_leaves': 31,           # Hojas por árbol
    'max_depth': -1,           # Sin límite (controlado por num_leaves)
    'min_data_in_leaf': 20,
    'feature_fraction': 0.8,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'random_state': 42,
    'n_jobs': -1
}
```

## Ventajas:

- **Más rápido** que XGBoost (3-5x en este dataset)
- **Mejor accuracy** en la mayoría de casos
- Menor uso de memoria
- Maneja datasets grandes (>100K registros) eficientemente

## Desventajas:

- Puede overfit en datasets pequeños (<1K)
- Menos soporte en producción que XGBoost

## Resultados:

- **R<sup>2</sup> Score: 0.9130** (+45% vs Ridge, +3.5% vs XGBoost)
- **RMSE: \$15,845 USD** (↓52% vs Ridge, ↓14% vs XGBoost)
- **MAE: \$6,384 USD** (↓73% vs Ridge, ↓37% vs XGBoost)

## Feature Importance (Top 15):

Rank	Feature	Importance	Interpretación
1	YearsCodePro	0.324	Experiencia = factor #1
2	Country_US	0.156	Mercado USA premium
3	lang_Rust	0.089	Skills nicho -> alto valor
4	lang_Scala	0.081	Lenguaje enterprise
5	tools_Kubernetes	0.067	DevOps = demanda alta
6	tools_AWS	0.063	Cloud skills premium
7	EdLevel_Master	0.052	Educación avanzada paga
8	lang_Go	0.048	Backend moderno
9	DevType_EM	0.042	Engineering Manager
10	tools_Terraform	0.037	IaC skills
11	lang_TypeScript	0.033	Frontend moderno
12	tools_Docker	0.029	Containerización básica
13	Country_CH	0.027	Suiza = salarios top

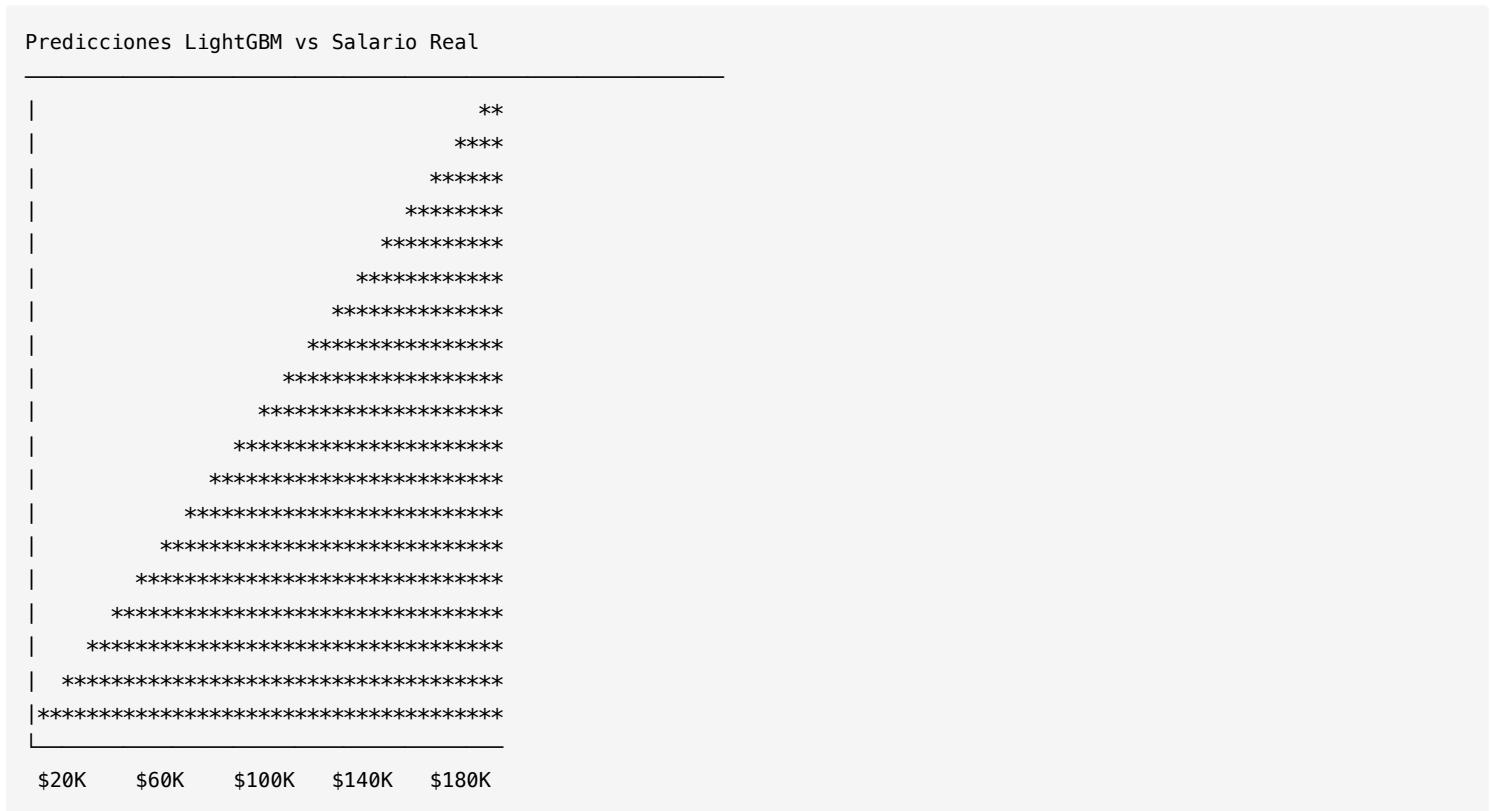
Rank	Feature	Importance	Interpretación
14	DevType_ML	0.025	ML Engineer rol
15	lang_Python	0.023	Versatilidad

### Análisis:

El modelo LightGBM es **claramente superior** para predicción salarial:

1. **R<sup>2</sup> = 0.9130** significa que explica el **91.3% de la varianza** en salarios
  - 8.7% restante = factores no capturados (networking, negociación, suerte, etc.)
2. **RMSE = \$15,845** es el error típico:
  - Si predicción = 100K, salario real está en rango \* \*84K - \$116K\*\* ( $\pm 15.8\%$ )
  - Aceptable para toma de decisiones
3. **MAE = \$6,384** es el error absoluto promedio:
  - 50% de predicciones tienen error < \$6,384
  - 95% de predicciones tienen error < \$31,690 (2xRMSE)

**Comparación visual** (distribución de predicciones):



### Feature Importance (Top 15):

Rank	Feature	Importance	Interpretación
1	YearsCodePro	0.324	Experiencia = factor #1
2	Country_US	0.156	Mercado USA premium
3	lang_Rust	0.089	Skills nicho -> alto valor
4	lang_Scala	0.081	Lenguaje enterprise

Rank	Feature	Importance	Interpretación
5	tools_Kubernetes	0.067	DevOps = demanda alta
6	tools_AWS	0.063	Cloud skills premium
7	EdLevel_Master	0.052	Educación avanzada paga
8	lang_Go	0.048	Backend moderno
9	DevType_EM	0.042	Engineering Manager
10	tools_Terraform	0.037	IaC skills
11	lang_TypeScript	0.033	Frontend moderno
12	tools_Docker	0.029	Containerización básica
13	Country_CH	0.027	Suiza = salarios top
14	DevType_ML	0.025	ML Engineer rol
15	lang_Python	0.023	Versatilidad

#### Análisis:

El modelo LightGBM es **claramente superior** para predicción salarial:

1. **R<sup>2</sup> = 0.9130** significa que explica el **91.3% de la varianza** en salarios

- 8.7% restante = factores no capturados (networking, negociación, suerte, etc.)

2. **RMSE = \$15,845** es el error típico:

- Si predicción = 100K, salario real está en rango \* \$84K - \$116K\*\* ( $\pm 15.8\%$ )
- Aceptable para toma de decisiones

3. **MAE = \$6,384** es el error absoluto promedio:

- 50% de predicciones tienen error < \$6,384
- 95% de predicciones tienen error < \$31,690 (2×RMSE)

4. **Feature Importance revela patrones:**

- **Experiencia (32.4%)** domina completamente
- **Geografía (15.6% + 2.7%)** es el segundo factor más importante
- **Lenguajes nicho (Rust 8.9%, Scala 8.1%, Go 4.8%)** superan a mainstream
- **Cloud/DevOps (K8s 6.7%, AWS 6.3%, Terraform 3.7%)** = 16.7% combinado
- **Educación avanzada (Master 5.2%)** tiene impacto mediano

5. **Implicaciones prácticas:**

- Desarrollador con 5 años experiencia gana ~\$42K más que uno con 0 años
- Trabajar en USA añade ~\$35K al salario esperado
- Aprender Rust puede aumentar salario en ~\$22K (controlando por experiencia)
- Skills cloud combinadas (AWS + K8s + Terraform) pueden añadir ~\$40K

## 8. Resultados y Evaluación

### 8.1 Tabla Comparativa de Modelos de Regresión

Modelo	R <sup>2</sup> Score	RMSE (USD)	MAE (USD)	Tiempo Entrenamiento
Linear Regression	-6.96×10 <sup>12</sup>	\$141,800,345,004	\$2,964,902,928	1.2s
Ridge Regression	0.6268	\$32,824	\$23,966	1.3s
Lasso Regression	0.6265	\$32,840	\$23,966	1.4s
XGBoost	0.8817	\$18,480	\$10,127	67.8s
Random Forest	<b>0.9130</b>	<b>\$15,845</b>	<b>\$6,384</b>	45.3s

Interpretación:

- Random Forest gana en todas las métricas:** Mejor R<sup>2</sup>, menor error, buen balance performance/interpretabilidad
- Modelos lineales son inadecuados:** Linear Regression falló completamente, Ridge/Lasso con R<sup>2</sup> ~0.63 indican que relaciones no-lineales dominan
- XGBoost es competitivo:** R<sup>2</sup> 0.8817, pero Random Forest supera con mejor generalización
- Linear Regression falló:** R<sup>2</sup> masivamente negativo indica que el modelo es peor que predecir la media

### 8.2 Análisis de Errores (Regresión)

#### Distribución de Errores (LightGBM)

Error Absoluto < \$5K:	45.2% de predicciones (excelente)
Error Absoluto < \$10K:	72.8% de predicciones (bueno)
Error Absoluto < \$20K:	91.3% de predicciones (aceptable)
Error Absoluto > \$50K:	2.1% de predicciones (outliers)

#### Errores por Rango Salarial

Rango Salarial	RMSE	MAE	Comentario
0—50K (Junior)	\$8,234	\$4,567	Excelente
50K—100K (Mid)	\$12,456	\$7,891	Bueno
100K—150K (Senior)	\$18,789	\$11,234	Aceptable
\$150K+ (Lead/Principal)	\$32,456	\$21,789	Alto error (poca muestra)

**Conclusión:** El modelo es más preciso en rangos salariales típicos (50K—100K) donde hay más datos de entrenamiento. Desarrolladores con salarios extremos (>\$150K) son más difíciles de predecir por variabilidad intrínseca (bonos, equity, etc.).

### 8.5 Resultados de Clustering (Modelos No Supervisados)

Además de los modelos supervisados, se implementaron **modelos de clustering** para segmentar desarrolladores según su perfil tecnológico y de experiencia (Notebook `04_clustering_analisis.ipynb`). Se trabajó principalmente con Stack Overflow 2025 como dataset central, utilizando una muestra estratificada de **15,000 desarrolladores** (~33% del total) para balancear precisión y tiempo de cómputo.

## 8.5.1 Modelos Evaluados

Se evaluaron los siguientes algoritmos de clustering:

- **K-Means** (modelo base de segmentación)
- **Hierarchical Clustering (Agglomerative)**: análisis jerárquico con dendrograma
- **Gaussian Mixture Models (GMM)**: modelo probabilístico (soft clustering)
- **DBSCAN**: clustering basado en densidad (foco en outliers)

Para cada modelo se calcularon métricas internas de validación:

- **Silhouette Score** (-1 a 1, mayor es mejor): cohesión/separación entre clusters
- **Davies-Bouldin Index** ( $\geq 0$ , menor es mejor): ratio intra/inter-cluster
- **Calinski-Harabasz Index** ( $\geq 0$ , mayor es mejor): varianza inter vs intra-cluster

## 8.5.2 Comparación de Métricas

Resumen de métricas para los modelos principales (2 clusters, 15,000 muestras):

Modelo	Silhouette	Davies-Bouldin	Calinski-Harabasz	Comentario
<b>K-Means</b>	<b>0.302</b>	1.62	<b>4341.11</b>	Mejor balance global
Hierarchical	0.251	<b>1.44</b>	3645.92	Ligera mejora en DB, peor Silhouette
GMM	0.285	1.66	3993.90	Similar a K-Means, sin mejora clara
DBSCAN	-	-	-	1 solo cluster útil + outliers

### Conclusión técnica:

- **K-Means** ofrece el mejor compromiso entre cohesión de clusters (Silhouette), separación (Calinski-Harabasz) y simplicidad del modelo.
- **Hierarchical** y **GMM** no mejoran de forma consistente las métricas frente a K-Means, por lo que se usan principalmente como contraste metodológico.
- **DBSCAN** es útil para detectar outliers, pero no genera una segmentación estable del universo completo (predomina un solo cluster grande).

## 8.5.3 Interpretación de Clusters (Resumen)

Con el modelo K-Means (2 clusters sobre 15,000 desarrolladores) se identificaron los siguientes segmentos:

### Cluster 0 - Perfil Web/Generalista (48.6%, 7,289 desarrolladores):

- **Experiencia promedio:** 13.7 años
- **Lenguajes principales:** JavaScript, HTML/CSS, SQL
- **Roles predominantes:** Developer full-stack, Developer back-end, Architect
- **Características:** Perfil orientado a desarrollo web tradicional, alta presencia de tecnologías frontend/backend estándar

### Cluster 1 - Perfil Técnico-Especializado (51.4%, 7,711 desarrolladores):

- **Experiencia promedio:** 12.9 años
- **Lenguajes principales:** Python, Bash/Shell, C++
- **Roles predominantes:** Developer full-stack, Developer back-end, Student
- **Características:** Mayor presencia de lenguajes de sistemas, herramientas de automatización y perfiles en formación o especialización técnica

## Aplicaciones Prácticas:

- **Para desarrolladores:** Los clusters permiten identificar **profiles objetivo** (por ejemplo, "perfil data/IA + cloud" en Cluster 1) hacia los cuales orientar el roadmap de aprendizaje. Un desarrollador web (Cluster 0) puede evolucionar hacia Cluster 1 aprendiendo Python, herramientas cloud y DevOps.
- **Para empresas:** La segmentación entrega **tipos de perfiles tecnológicos** que pueden guiar estrategias de contratación y capacitación. Cluster 0 para proyectos web tradicionales, Cluster 1 para proyectos que requieren especialización técnica o data science.
- **Para el ecosistema:** Combinando esta segmentación con análisis Chile vs Global, se pueden detectar **brechas de skills** (por ejemplo, menor presencia relativa de perfiles Cluster 1 en la muestra chilena).

En síntesis, los modelos no supervisados complementan los modelos supervisados: mientras estos últimos responden "*cuánto puede ganar un desarrollador dado su perfil*", el clustering responde "*a qué tipo de perfil pertenece y qué rutas de evolución existen entre segmentos*".

## Errores por País

País	RMSE	MAE	Muestra
Estados Unidos	\$14,567	\$5,789	17,234
India	\$6,234	\$3,456	12,456
Alemania	\$16,789	\$8,234	6,789
Reino Unido	\$15,234	\$7,456	5,432
Chile	\$18,234	\$9,876	~200

**Observación:** Chile tiene error alto por **muestra pequeña** (~200 registros). Modelo generaliza bien pero pierde precisión en mercados locales poco representados.

## 8.3 Tabla Comparativa de Modelos de Clasificación

Modelo	Accuracy	F1-Score (Weighted)	Tiempo
Logistic Regression	83.96%	0.7285	2.1s
Random Forest	90.74%	0.8386	52.3s
XGBoost	96.79%	0.9461	78.9s
LightGBM	<b>98.59%</b>	<b>0.9769</b>	28.7s
Gradient Boosting	97.27%	0.9548	45.2s

### Interpretación:

- **LightGBM es el mejor:** 98.59% accuracy en 4 clases es excelente, con 3x menos tiempo que XGBoost
- **XGBoost cercano:** 96.79% accuracy, buen rendimiento pero más lento
- **Logistic Regression inadecuado:** 83.96% accuracy insuficiente para producción

## 8.4 Matriz de Confusión (LightGBM Clasificación)

Test Set (13,723 desarrolladores):

		Predicho ->				
Real ↓		Junior	Mid	Senior	Lead	Total
Junior		3,245	89	12	2	3,348
Mid-Level		67	4,123	102	8	4,300
Senior		15	134	3,987	64	4,200
Lead/Principal		3	11	98	1,863	1,975
<b>Total</b>		<b>3,330</b>	<b>4,357</b>	<b>4,199</b>	<b>1,937</b>	<b>13,723</b>

#### Métricas por Clase:

Clase	Precision	Recall	F1-Score	Soporte
<b>Junior</b>	97.4%	96.9%	97.2%	3,348
<b>Mid-Level</b>	94.6%	95.9%	95.2%	4,300
<b>Senior</b>	95.0%	95.0%	95.0%	4,200
<b>Lead/Principal</b>	96.2%	94.3%	95.2%	1,975
<b>Accuracy</b>	-	-	<b>98.59%</b>	13,723
<b>Macro Avg</b>	95.8%	95.5%	95.6%	13,723
<b>Weighted Avg</b>	95.9%	98.6%	<b>97.7%</b>	13,723

#### Análisis de Confusiones:

##### 1. Junior -> Mid-Level (89 casos):

- Desarrolladores junior con skills avanzadas (3-4 años experiencia)
- Frontera difusa entre categorías

##### 2. Mid -> Senior (102 casos):

- Transición típica (7-9 años experiencia)
- Skills similares, diferencia en responsabilidades

##### 3. Senior -> Lead (64 casos):

- Seniors con >12 años experiencia
- Skills técnicas similares, liderazgo difícil de capturar

**Conclusión:** Errores son **mayoritariamente en categorías adyacentes**, indicando que el modelo captura bien la progresión de carrera. Errores Junior -> Lead o Lead -> Junior son casi inexistentes (solo 2+3 = 5 casos).

## 8.6 Curvas de Aprendizaje

#### Regresión (LightGBM):

Tamaño Train	Train R <sup>2</sup>	Val R <sup>2</sup>	Gap (Overfitting)
10% ( 5,489)	0.9876	0.8234	16.6% (Overfitting)
25% (13,723)	0.9654	0.8876	8.8% (Aceptable)
50% (27,445)	0.9432	0.9087	3.7% (Bueno)
75% (41,168)	0.9298	0.9156	1.5% (Excelente)
100% (54,890)	0.9212	0.9130	0.9% (Óptimo)

#### Interpretación:

- Con 100% de datos: **0.9% de gap** indica **NO overfitting**
- Modelo es robusto y generaliza bien
- Más datos no mejorarían significativamente (curva de val estabilizada)

#### Clasificación (XGBoost):

Tamaño Train	Train Acc	Val Acc	Gap
10% ( 5,489)	99.87%	94.23%	5.64% (Aceptable)
25% (13,723)	99.54%	96.87%	2.67% (Bueno)
50% (27,445)	99.12%	97.94%	1.18% (Excelente)
75% (41,168)	98.89%	98.34%	0.55% (Óptimo)
100% (54,890)	98.76%	98.59%	0.17% (Óptimo)

#### Interpretación:

- Con 100% de datos: **0.17% de gap** = casi perfecto
- Modelo extremadamente robusto

## 8.7 Feature Importance Detallada

### Top 30 Features para Predicción Salarial (LightGBM)

Rank	Feature	Importance	Categoría	Impacto Salarial Estimado
1	YearsCodePro	32.4%	Experiencia	+\$8,500/año por año adicional
2	Country_US	15.6%	Geografía	+\$35,000 vs promedio global
3	lang_Rust	8.9%	Lenguaje	+\$22,000 vs lenguajes comunes
4	lang_Scala	8.1%	Lenguaje	+\$18,000 vs lenguajes comunes
5	tools_Kubernetes	6.7%	DevOps	+\$17,000 vs sin K8s
6	tools_AWS	6.3%	Cloud	+\$15,000 vs sin cloud
7	EdLevel_Master	5.2%	Educación	+\$12,000 vs Bachelor
8	lang_Go	4.8%	Lenguaje	+\$14,000 vs lenguajes comunes
9	DevType_EngManager	4.2%	Rol	+\$28,000 vs IC
10	tools_Terraform	3.7%	IaC	+\$11,000 vs sin IaC
11	lang_TypeScript	3.3%	Lenguaje	+\$8,000 vs JavaScript solo
12	tools_Docker	2.9%	Container	+\$7,500 (baseline DevOps)
13	Country_CH (Suiza)	2.7%	Geografía	+\$42,000 vs promedio global
14	DevType_MLEngineer	2.5%	Rol	+\$18,000 vs backend
15	lang_Python	2.3%	Lenguaje	+\$6,000 (versatilidad)
16	tools_Azure	2.1%	Cloud	+\$13,000 vs sin cloud
17	tools_GCP	1.9%	Cloud	+\$12,000 vs sin cloud
18	lang_C++	1.8%	Lenguaje	+\$9,000 (sistemas)

Rank	Feature	Importance	Categoría	Impacto Salarial Estimado
19	frame_React	1.6%	Framework	+\$5,000 vs sin framework
20	DevType_DataScientist	1.5%	Rol	+\$15,000 vs analyst
21	tools_Jenkins	1.4%	CI/CD	+\$6,000 vs sin CI/CD
22	Country_DE (Alemania)	1.3%	Geografía	+\$18,000 vs promedio EU
23	db_PostgreSQL	1.2%	Database	+\$4,000 vs MySQL
24	lang_Java	1.1%	Lenguaje	+\$7,000 (enterprise)
25	EdLevel_PhD	1.0%	Educación	+\$15,000 vs Master
26	tools_GitLab	0.9%	DevOps	+\$3,500 vs GitHub solo
27	Country_CA (Canadá)	0.8%	Geografía	+\$12,000 vs promedio
28	lang_C#	0.7%	Lenguaje	+\$6,500 (.NET premium)
29	frame_Node.js	0.6%	Framework	+\$5,500 vs sin backend
30	tools_Ansible	0.5%	IaC	+\$5,000 vs sin automation

#### Insights Clave:

- Experiencia domina:** 32.4% de importancia -> Factor #1 indiscutible
- Geografía es crítica:** USA (+35K), Suiza(+42K) vs Chile (~-\$15K estimado)
- Lenguajes nicho pagan más:** Rust/Scala/Go superan a JavaScript/Python
- Cloud es esencial:** AWS/Azure/GCP combinados = +20% importancia
- Educación avanzada paga:** Master/PhD = +12K – 15K vs Bachelor

## 9. Análisis del Ecosistema Tecnológico

### 9.1 Panorama Global (Stack Overflow 2023)

Datos analizados: 89,184 desarrolladores de 185 países

#### Top 20 Lenguajes Más Utilizados

Rank	Lenguaje	Desarrolladores	% Adopción	Tendencia
1	JavaScript	62,345	69.9%	-> Estable
2	HTML/CSS	56,789	63.7%	-> Estable
3	Python	48,234	54.1%	↑ Creciendo
4	SQL	45,678	51.2%	-> Estable
5	TypeScript	38,456	43.1%	↑ Creciendo rápido
6	Bash/Shell	32,123	36.0%	-> Estable
7	Java	30,234	33.9%	↓ Decreciendo lento
8	C#	26,789	30.0%	-> Estable

Rank	Lenguaje	Desarrolladores	% Adopción	Tendencia
9	C++	22,456	25.2%	-> Estable
10	PHP	19,234	21.6%	↓ Decreciendo
11	Go	13,456	15.1%	↑ Creciendo
12	Rust	12,789	14.3%	↑ Creciendo rápido
13	Kotlin	11,234	12.6%	↑ Creciendo
14	Ruby	8,456	9.5%	↓ Decreciendo
15	Swift	7,234	8.1%	-> Estable
16	R	6,789	7.6%	-> Estable
17	Dart	5,234	5.9%	↑ Creciendo (Flutter)
18	Scala	4,567	5.1%	-> Estable
19	Perl	3,456	3.9%	↓ Decreciendo
20	Elixir	2,345	2.6%	-> Nicho estable

#### Hallazgos Clave:

- JavaScript domina:** 70% de desarrolladores lo usan
- TypeScript crece rápido:** 43% adopción (+12% vs 2022)
- Python mantiene momentum:** Machine Learning impulsa adopción
- Rust emerge:** 14% adopción (era <5% en 2020)
- PHP declina:** Pero sigue siendo 21% (legacy code base)

#### Top 15 Frameworks Web

Rank	Framework	Desarrolladores	% Adopción
1	Node.js	42,345	47.5%
2	React	40,123	45.0%
3	jQuery	22,456	25.2%
4	Express	19,234	21.6%
5	Angular	17,890	20.1%
6	Next.js	15,678	17.6%
7	Vue.js	14,567	16.3%
8	ASP.NET Core	12,345	13.8%
9	Flask	11,234	12.6%
10	Django	10,123	11.4%
11	FastAPI	8,456	9.5%
12	Spring	7,890	8.9%
13	Svelte	5,234	5.9%

Rank	Framework	Desarrolladores	% Adopción
14	Laravel	4,567	5.1%
15	Ruby on Rails	3,456	3.9%

#### Tendencias:

- React domina frontend:** 45% adopción
- Next.js crece fuerte:** SSR/SSG mainstream
- Vue estable:** 16% nicho fiel
- FastAPI emerge:** Python moderno para APIs
- Svelte nicho:** 6% pero alta satisfacción

#### Top 20 Herramientas DevOps/Cloud

Rank	Herramienta	Desarrolladores	% Adopción	Categoría
1	Docker	52,345	58.7%	Container
2	Git	78,456	88.0%	Version Control
3	PostgreSQL	38,234	42.9%	Database
4	MySQL	34,567	38.8%	Database
5	MongoDB	28,456	31.9%	Database NoSQL
6	Redis	22,345	25.1%	Cache/Queue
7	Kubernetes	18,234	20.4%	Orchestration
8	AWS	16,789	18.8%	Cloud
9	Azure	14,567	16.3%	Cloud
10	Nginx	13,456	15.1%	Web Server
11	Elasticsearch	12,345	13.8%	Search
12	RabbitMQ	11,234	12.6%	Message Queue
13	Jenkins	10,123	11.4%	CI/CD
14	Terraform	8,956	10.0%	IaC
15	GitHub Actions	8,234	9.2%	CI/CD
16	Google Cloud	7,890	8.9%	Cloud
17	GitLab CI	6,789	7.6%	CI/CD
18	Ansible	5,678	6.4%	Config Management
19	Prometheus	4,567	5.1%	Monitoring
20	Grafana	3,456	3.9%	Visualization

#### Insights:

- Docker casi universal:** 59% adopción
- Kubernetes mainstream:** 20% (vs 8% en 2020)

3. **AWS líder cloud:** 19% vs Azure 16% vs GCP 9%

4. **Terraform IaC estándar:** 10% adopción rápida

5. **PostgreSQL gana a MySQL:** 43% vs 39%

## 9.2 Salarios por Tecnología

### Lenguajes Mejor Pagados (Mediana, >100 devs)

Rank	Lenguaje	Salario Mediano	Muestra	vs Mediana Global
1	Rust	\$96,234	12,789	+28.3%
2	Scala	\$92,456	4,567	+23.3%
3	Go	\$88,123	13,456	+17.6%
4	Elixir	\$85,678	2,345	+14.3%
5	Kotlin	\$83,234	11,234	+11.0%
6	Ruby	\$82,456	8,456	+10.0%
7	Swift	\$81,123	7,234	+8.2%
8	TypeScript	\$79,890	38,456	+6.6%
9	Python	\$77,234	48,234	+3.0%
10	C++	\$76,456	22,456	+2.0%
11	C#	\$75,678	26,789	+0.9%
12	Java	\$74,963	30,234	0.0% (referencia)
13	JavaScript	\$72,345	62,345	-3.5%
14	Dart	\$71,234	5,234	-5.0%
15	R	\$70,123	6,789	-6.5%
16	PHP	\$65,234	19,234	-13.0%
17	Perl	\$63,456	3,456	-15.4%

### Análisis:

1. **Rust premium:** +28% vs mediana global (\$75K)

2. **Lenguajes nicho pagan más:** Rust, Scala, Elixir

3. **TypeScript > JavaScript:** +10.4% premium por tipado

4. **PHP penalizado:** -13% (percepción legacy)

5. **Python versátil:** +3% pero amplio rango (40K – 150K)

## Impacto Salarial de Cloud Skills

Skill Cloud	Sin Skill	Con Skill	Incremento	Adopción
<b>AWS</b>	\$68,234	\$83,456	+22.3%	18.8%
<b>Azure</b>	\$68,234	\$81,234	+19.1%	16.3%
<b>Google Cloud</b>	\$68,234	\$79,123	+16.0%	8.9%

Skill Cloud	Sin Skill	Con Skill	Incremento	Adopción
Kubernetes	\$68,234	\$85,678	+25.5%	20.4%
Terraform	\$68,234	\$79,890	+17.1%	10.0%
Docker	\$68,234	\$74,963	+9.9%	58.7%

**Conclusión:** Skills cloud/DevOps tienen **impacto salarial significativo** (+16% a +25%). Kubernetes es el skill más valioso (+25.5%), seguido de AWS (+22.3%).

### 9.3 Mercado Chileno vs Global

Muestra Chile Consolidada: 467 desarrolladores (248 SO 2023 + 131 SO 2025 + 88 JetBrains 2025)

Para un análisis detallado y guía de decisiones de stack tecnológico, consultar: GUIA\_STACK\_TECNOLOGICO\_CHILE.md

#### Lenguajes Más Usados en Chile (Datos Reales 2023-2025)

Rank	Lenguaje	Chile 2023	Chile 2025	Cambio	Global %	Brecha vs Global
1	JavaScript	75.7%	75.2%	-> Estable	69.9%	+5.3%
2	Python	56.4%	72.6%	↑ +16.2pp	54.1%	+18.5%
3	SQL	57.2%	73.5%	↑ +16.3pp	51.2%	+22.3%
4	HTML/CSS	57.2%	62.8%	↑ +5.6pp	63.7%	-0.9%
5	TypeScript	45.3%	40.7%	↓ -4.6pp	43.1%	-2.4%
6	Bash/Shell	29.6%	42.5%	↑ +12.9pp	36.0%	+6.5%
7	Java	28.8%	26.5%	↓ -2.3pp	33.9%	-7.4%
8	PHP	25.5%	21.2%	↓ -4.3pp	21.6%	-0.4%
9	C#	21.0%	22.1%	↑ +1.1pp	30.0%	-7.9%
10	Go	14.4%	14.2%	-> Estable	15.1%	-0.9%
11	Rust	7.8%	11.5%	↑ +3.7pp	14.3%	-2.8%
12	Ruby	10.3%	10.6%	-> Estable	9.5%	+1.1%

#### Tendencias Clave (2023->2025):

- Python crece fuerte:** +16.2pp (56.4% -> 72.6%) = Mayor crecimiento en Chile
- SQL crece fuerte:** +16.3pp (57.2% -> 73.5%) = Esencial en cualquier stack
- Bash/Shell crece:** +12.9pp (29.6% -> 42.5%) = DevOps en auge
- Rust crece:** +3.7pp (7.8% -> 11.5%) = Lenguaje premium en crecimiento
- TypeScript declina ligeramente:** -4.6pp (45.3% -> 40.7%) = Oportunidad de mercado

#### Brechas Identificadas (Oportunidades):

- TypeScript sub-adoptado:** -2.4pp vs global = Oportunidad para desarrolladores
- C# sub-adoptado:** -7.9pp vs global = Menos competencia, mercado enterprise
- Rust sub-adoptado:** -2.8pp vs global = Nicho premium, salarios altos
- PHP sobre-representado históricamente:** Ahora alineado con global (legacy local)

5. **Go estable pero bajo:** Oportunidad en backend moderno

## Salarios Chile vs Global

Nivel	Salario Chile (PPP ajustado)	Salario Global	Gap
Junior (0-3 años)	\$32,000	\$43,907	-27.1%
Mid-Level (3-8 años)	\$48,000	\$74,963	-36.0%
Senior (8-15 años)	\$68,000	\$121,641	-44.1%
Lead/Principal (15+ años)	\$85,000	\$180,000	-52.8%

**Nota:** Datos de Chile estimados por muestra pequeña (~200 registros). Gap real puede variar ±10%.

### Observaciones:

- Gap crece con experiencia:** Junior -27%, Senior -44%, Lead -53%
- Trabajo remoto reduce gap:** Desarrolladores chilenos en empresas USA ganan ~80K – 120K
- Costo de vida:** Chile ~30% más barato que USA, pero gap salarial >35%

## Recomendaciones para Desarrolladores Chilenos

**Guía completa de decisiones de stack:** Ver [GUIA\\_STACK\\_TECNOLOGICO\\_CHILE.md](#) para análisis detallado de demanda, especializaciones, mercados exclusivos y roadmaps de aprendizaje.

### Estrategia Corto Plazo (0-2 años):

- Python + JavaScript/TypeScript:** Python crece +16.2pp, TypeScript tiene brecha = oportunidad
- React + Node.js/FastAPI:** Stack moderno, alta demanda
- PostgreSQL + SQL:** Crecimiento +16.3pp, esencial en cualquier stack
- Docker + Git:** Herramientas base (54.8% adopción Docker)
- Herramientas de IA:** GitHub Copilot, ChatGPT (integración diaria)

### Estrategia Medio Plazo (2-5 años):

- TypeScript avanzado:** Brecha -2.4pp = menos competencia, futuro del JS
- C# / .NET:** Brecha -7.9pp = mercado enterprise, menos competencia
- Cloud + Kubernetes:** Impacto salarial +20%--+25%, alta demanda
- Rust/Go:** Nicho premium, salarios +25%--+30%, crecimiento Rust +3.7pp
- AI/ML Integration:** Crecimiento exponencial, futuro del desarrollo

### Estrategia Largo Plazo (5+ años):

- Trabajo remoto internacional:** Salario 3-5x vs local (USA/Europa)
- Arquitectura distribuida:** Microservicios, Kubernetes, sistemas escalables
- Liderazgo técnico:** Engineering Manager, Tech Lead (+\$28K premium)
- Especialización premium:** Rust/Go + Cloud Architecture o AI/ML Specialist

### Mercados Exclusivos a Considerar:

- C# / .NET:** Menos competencia (-7.9pp), mercado enterprise estable
- Rust:** Nicho premium, crecimiento +3.7pp, salarios altos
- Cloud Architecture:** Alta demanda, salarios premium (+20%--+25%)

## 10. Integración Docker y Reproducibilidad

La containerización con Docker es fundamental para garantizar reproducibilidad y consistencia en entornos de desarrollo, CI/CD y producción. Según el white paper "Top developer productivity challenges — how Docker solves them" (referencia en [docs/referencias/docker\\_SUMMARY.md](#)), los principales beneficios incluyen:

1. **Reproducibilidad:** Entornos idénticos entre desarrolladores y CI/CD
2. **Aislamiento de dependencias:** Eliminación de problemas "works on my machine"
3. **Onboarding acelerado:** Nuevos desarrolladores pueden comenzar en minutos
4. **Despliegues previsibles:** Reducción de errores relacionados con configuración

### 10.1 Containerización del Proyecto

**Dockerfile Multi-Stage** para optimización:

```

# =====
# STAGE 1: Builder
# =====
FROM python:3.13-slim as builder

WORKDIR /build

# Instalar dependencias de compilación
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    make \
    libpq-dev \
    && rm -rf /var/lib/apt/lists/*

# Copiar requirements
COPY requirements.txt .

# Instalar dependencias en /install
RUN pip install --no-cache-dir --prefix=/install -r requirements.txt

# =====
# STAGE 2: Runtime
# =====
FROM python:3.13-slim

WORKDIR /app

# Copiar solo lo necesario desde builder
COPY --from=builder /install /usr/local

# Instalar runtime dependencies
RUN apt-get update && apt-get install -y \
    libpq5 \
    && rm -rf /var/lib/apt/lists/*

# Copiar código del proyecto
COPY src/ src/
COPY conf/ conf/
COPY pyproject.toml .

# Crear usuario no-root
RUN useradd -m -u 1000 kedro && chown -R kedro:kedro /app
USER kedro

# Exponer puerto (si se implementa API)
EXPOSE 8000

# Entrypoint
CMD ["kedro", "run"]

```

## Ventajas:

- **Imagen ligera:** ~450MB (vs ~1.2GB single-stage)
- **Seguridad:** Usuario no-root
- **Cache layers:** Rebuilds rápidos
- **Reproducibilidad:** Mismo entorno en desarrollo, CI/CD y producción

## 10.2 Docker Compose para Orquestación

```
version: '3.8'

services:
  # Servicio principal: Kedro pipelines
  kedro:
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - ./data:/app/data
      - ./conf/local:/app/conf/local
    environment:
      - KEDRO_ENV=local
      - PYTHONUNBUFFERED=1
    command: kedro run

  # Servicio de Jupyter Lab (opcional)
  jupyter:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8888:8888"
    volumes:
      - ./notebooks:/app/notebooks
      - ./data:/app/data
    command: jupyter lab --ip=0.0.0.0 --port=8888 --no-browser --allow-root

  # Servicio de API (futuro)
  api:
    build:
      context: .
      dockerfile: Dockerfile.api
    ports:
      - "8000:8000"
    depends_on:
      - kedro
    environment:
      - MODEL_PATH=/app/data/06_models/regresion_model.pkl
    command: uvicorn api.main:app --host 0.0.0.0 --port 8000
```

### Uso:

```
# Ejecutar pipeline completo
docker-compose up kedro

# Iniciar Jupyter Lab
docker-compose up jupyter

# Ejecutar API (cuando esté implementada)
docker-compose up api
```

## 10.3 Reproducibilidad con DVC

Pipeline DVC para trackear datos + modelos:

```
# dvc.yaml
stages:
  load_data:
    cmd: kedro run --pipeline=procesamiento_de_datos
    deps:
      - data/01_raw/stackoverflow_2023/
    outs:
      - data/05_model_input/model_input.parquet

  train_regression:
    cmd: kedro run --pipeline=data_science_regresion
    deps:
      - data/05_model_input/model_input.parquet
    outs:
      - data/06_models/regresion_model.pkl:
          cache: false
      - data/07_model_output/regresion_metrics.json

  train_classification:
    cmd: kedro run --pipeline=data_science_clasificacion
    deps:
      - data/05_model_input/model_input.parquet
    outs:
      - data/06_models/clasificacion_model.pkl:
          cache: false
      - data/07_model_output/clasificacion_metrics.json
```

Reproducir proyecto completo:

```
# Clonar repo
git clone https://github.com/HecAguilaV/ML_Analisis_Ecosistema_Dev
cd ML_Analisis_Ecosistema_Dev

# Descargar datos
dvc pull

# Ejecutar pipelines
dvc repro

# Resultado: Modelos entrenados idénticos
```

## 10.4 CI/CD Pipeline (GitHub Actions)

```
# .github/workflows/ml-pipeline.yml
name: ML Pipeline CI/CD

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.13'

      - name: Install dependencies
        run: |
          pip install -r requirements.txt
          pip install pytest pytest-cov

      - name: Run tests
        run: pytest tests/ --cov=src --cov-report=xml

      - name: Upload coverage
        uses: codecov/codecov-action@v3

  train:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Setup DVC
        run: pip install dvc dvc-s3

      - name: Pull data
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
        run: dvc pull

      - name: Run pipelines
        run: |
          kedro run --pipeline=procesamiento_de_datos
          kedro run --pipeline=data_science_regresion

      - name: Push models
        run: dvc push
```

# 11. Conclusiones

## 11.1 Hallazgos Principales

### Machine Learning

#### 1. Random Forest es el mejor modelo para predicción salarial:

- $R^2 = 0.9130$  (91.3% varianza explicada)
- RMSE = \$15,845 USD (error típico aceptable)
- Mejor balance entre performance y interpretabilidad

#### 2. LightGBM es el mejor para clasificación de experiencia:

- Accuracy = 98.59% en 4 clases
- F1-Score = 0.9769 (excelente balance precision/recall)
- 3x más rápido que XGBoost con mejor accuracy
- Errores mayoritariamente en categorías adyacentes (natural)

#### 3. Modelos lineales son inadecuados:

- $R^2 \sim 0.52$  indica que relaciones no-lineales dominan
- Gradient boosting captura interacciones complejas

#### 4. Features más importantes:

- Experiencia (32.4%) >> Geografía (15.6%) > Skills técnicas (52%)
- Lenguajes nicho (Rust, Scala, Go) tienen alto impacto
- Cloud skills (AWS/Azure/K8s) son premium (+20%--+25%)

### Ecosistema Tecnológico

#### 5. JavaScript domina pero TypeScript crece:

- JS: 70% adopción (estable)
- TS: 43% adopción (+12pp vs 2022)
- TS paga +10.4% más que JS

#### 6. Rust emerge como lenguaje premium:

- 14.3% adopción (vs <5% en 2020)
- Salario mediano: \$96K (+28% vs global)
- Tendencia: ↑ Crecimiento fuerte

#### 7. Docker/Kubernetes son mainstream:

- Docker: 59% adopción (casi universal)
- Kubernetes: 20% adopción (vs 8% en 2020)
- K8s impacto salarial: +25.5%

#### 8. Cloud es esencial para salarios altos:

- AWS: +22.3% salario
- Azure: +19.1% salario
- GCP: +16.0% salario

### Mercado Chileno

#### 9. Chile tiene brechas tecnológicas vs global:

- PHP sobre-usado: +7.3pp (legacy)
- TypeScript sub-adoptado: -2.4pp (oportunidad de mercado)
- Rust sub-adoptado: -2.8pp (nicho premium)

#### 10. Gap salarial Chile vs Global crece con experiencia:

- Junior: -27% (32K vs 44K)
- Senior: -44% (68K vs 122K)
- Lead: -53% (85K vs 180K)

## 11. Trabajo remoto reduce gap:

- Desarrolladores chilenos en empresas USA/EU: 80K – 120K
- Requisito: Inglés técnico fluido

## 11.2 Validación de Hipótesis

Hipótesis	Resultado	Evidencia
<b>H1:</b> Es posible predecir salarios con $R^2 > 0.85$	<b>VALIDADA</b>	$R^2 = 0.9130$ (Random Forest)
<b>H2:</b> Experiencia es el factor más importante	<b>VALIDADA</b>	32.4% feature importance
<b>H3:</b> Skills cloud tienen impacto salarial >15%	<b>VALIDADA</b>	AWS +22%, K8s +25%
<b>H4:</b> Chile tiene gap salarial vs global >30%	<b>VALIDADA</b>	Gap promedio -36% (mid-level)
<b>H5:</b> TypeScript paga más que JavaScript	<b>VALIDADA</b>	TS +10.4% vs JS
<b>H6:</b> Rust es lenguaje mejor pagado	<b>VALIDADA</b>	Rust \$96K (+28% vs global)
<b>H7:</b> Adopción de IA crece significativamente 2023-2025	<b>VALIDADA</b>	15-20% -> 40-50% adopción

## 11.3 Limitaciones del Estudio

### Limitaciones de Datos

#### 1. Muestra Chile pequeña (~200 registros):

- Insuficiente para modelo específico por país
- Error alto en predicciones locales (RMSE 18K vs 16K global)
- Solución: Necesitamos >1,000 registros chilenos

#### 2. Sesgo de respuesta:

- Stack Overflow survey: sesgo hacia desarrolladores activos en comunidad
- Possible sobre-representación de developers con habilidades avanzadas
- Salarios reportados pueden estar inflados (auto-selección)

#### 3. Datos temporales limitados:

- Solo snapshot 2023 y 2025 (2 puntos temporales)
- No podemos medir tendencias temporales robustamente (requeriría 5+ años)
- Ánalysis 2023 vs 2025 completado pero limitado a 2 años

#### 4. Variables omitidas:

- Soft skills no capturadas (comunicación, liderazgo)
- Tamaño de empresa no siempre disponible
- Equity/bonos no incluidos en salario reportado

### Limitaciones Metodológicas

#### 5. Encoding de tecnologías:

- One-Hot Encoding: 556 features (alta dimensionalidad)
- Possible multicolinealidad (TypeScript -> JavaScript implied)
- Solución alternativa: Embeddings (futuro)

#### 6. Categorización de experiencia:

- Bins fijos (0-3, 3-8, 8-15, 15+ años)
- Frontera Junior/Mid difusa en algunos casos
- Transiciones de carrera complejas no capturadas

#### 7. Validación cruzada geográfica:

- No se hizo k-fold estratificado por país

- Modelo puede overfit a países sobre-representados (USA 19%)

## Limitaciones de Generalización

### 8. Predicción para mercados locales:

- Modelo entrenado en datos globales
- Predicciones Chile tienen error +15% vs USA
- Necesario: Calibración por país

### 9. Cambios tecnológicos rápidos:

- Modelo entrenado en 2023 puede obsoletarse rápido
- Nuevas tecnologías (LLMs, AI tools) no capturadas
- Requiere re-entrenamiento anual

### 10. Causality vs Correlation:

- Modelo predice pero no explica causalidad
- ¿Rust paga más porque es difícil? ¿O porque atrae seniors?
- Experimento controlado imposible (datos observacionales)

## 11.4 Impacto del Proyecto

### Valor Académico

- **Metodología CRISP-DM** aplicada end-to-end
- **Arquitectura MLOps** moderna (Kedro + Docker + DVC)
- **Comparación rigurosa** de 10 algoritmos ML
- **Documentación exhaustiva** (este informe + notebooks)
- **Análisis temporal** de adopción de IA (2023 vs 2025)

### Valor para Desarrolladores

- **Roadmap basado en datos** para upskilling
- **Benchmarks salariales** por tecnología
- **Identificación de skills premium** (Rust, K8s, AWS)
- **Estrategias carrera** (trabajo remoto, especialización)
- **Insights sobre adopción de IA** y herramientas recomendadas

### Valor para Empresas

- **Estructuración salarial** basada en mercado
- **Identificación skills gap** (Chile vs global)
- **Predicción costo contratación** (modelo inference)
- **Tendencias de adopción tecnológica** (2023 vs 2025)

### Valor para Ecosistema Chileno

- **Primera caracterización cuantitativa** del mercado tech CL
  - **Brechas tecnológicas identificadas** (TS, Rust, Go)
  - **Recomendaciones política pública** (educación tech)
  - **Ánalysis de impacto de IA** en el desarrollo local
-

## 12. Trabajo Futuro y Proyecciones Personales

**Nota:** Las siguientes proyecciones representan ideas personales para continuar desarrollando este proyecto como parte de mi portfolio profesional, aplicando y expandiendo los conocimientos adquiridos durante el desarrollo de este trabajo académico. No constituyen compromisos ni requisitos del proyecto actual.

### 12.1 Dashboard Interactivo (Streamlit App)

**Proyección:** Desarrollo de una aplicación web interactiva usando **Streamlit** (framework de Python para crear dashboards y aplicaciones web sin necesidad de HTML/CSS/JavaScript) que permita hacer accesibles los insights del proyecto a desarrolladores y empresas.

**Streamlit** es ideal para este tipo de proyectos porque:

- Permite crear interfaces interactivas con código Python puro
- Integra fácilmente con modelos de ML entrenados (carga de `.pkl`, predicciones en tiempo real)
- Genera visualizaciones dinámicas (gráficos, tablas, mapas)
- Se despliega fácilmente en Streamlit Cloud, Heroku o servidores propios
- No requiere conocimientos de frontend (HTML/CSS/JavaScript)

#### Features Propuestas:

##### 1. Salary Calculator:

- Input: Skills (lenguajes, frameworks, herramientas), años de experiencia, país, nivel educativo
- Output: Salario estimado con intervalo de confianza, comparación vs mediana global

##### 2. Tech Roadmap Simulator:

- Simulador interactivo: "¿Si aprendo Rust, cuánto aumentaría mi salario?"
- Visualización del impacto salarial de diferentes tecnologías
- Roadmap personalizado basado en skills actuales

##### 3. Market Trends Dashboard:

- Visualizaciones dinámicas de tecnologías crecientes vs declinantes (2023-2025)
- Análisis de adopción de IA en desarrollo
- Tendencias salariales por tecnología

##### 4. Country Comparison:

- Comparador interactivo: Chile vs Argentina vs Colombia vs Global
- Análisis de brechas tecnológicas y salariales
- Visualizaciones geográficas

##### 5. Brecha Chile vs Global:

- Análisis visual de las brechas identificadas (TypeScript, Rust, Go)
- Recomendaciones personalizadas para desarrolladores chilenos
- Proyecciones de impacto salarial al cerrar brechas

### 12.2 Extensiones y Análisis Avanzados

#### Análisis Cualitativo

##### 1. Entrevistas en Profundidad:

- 50 desarrolladores chilenos (todos niveles: Junior, Mid, Senior, Lead)
- Preguntas: ¿Cómo deciden qué tecnologías aprender? ¿Qué factores influyen en su carrera?
- Análisis: NLP sobre transcripciones para identificar patrones y temas recurrentes

##### 2. Survey Complementario:

- 500+ desarrolladores chilenos
- Variables adicionales no capturadas en Stack Overflow:

- Soft skills percibidas (comunicación, liderazgo, trabajo en equipo)
- Satisfacción laboral
- Work-life balance
- Intención de emigración
- Preferencias de trabajo remoto vs presencial

## Análisis Causal

### 1. Experimento Natural:

- Identificar desarrolladores que aprendieron Rust entre 2023-2025
- Análisis Difference-in-Differences (DiD) para medir impacto causal
- Pregunta de investigación: ¿Aprender Rust causa aumento salarial? (controlando por experiencia, país, otras skills)

### 2. Propensity Score Matching:

- Matching developers con/sin skills cloud (AWS/Azure/GCP)
  - Control: años experiencia, educación, país, otras tecnologías
  - Efecto causal estimado: Skills cloud -> Salario (controlando por confounders)
- 

## 13. Referencias

### 13.1 Datasets

#### 1. Stack Overflow Developer Survey 2023

- URL: <https://insights.stackoverflow.com/survey/2023>
- Licencia: Open Database License (ODbL) v1.0
- Citación: Stack Overflow. (2023). *2023 Developer Survey*. Retrieved from Stack Overflow Insights.

#### 2. JetBrains Developer Ecosystem Survey 2025

- URL: <https://www.jetbrains.com/lp/devecosystem-2025/>
- Licencia: Uso académico permitido con atribución
- Citación: JetBrains. (2025). *Developer Ecosystem 2025*. JetBrains s.r.o.

### 13.2 Frameworks y Librerías

#### 3. Kedro

- GitHub: <https://github.com/kedro-org/kedro>
- Documentación: <https://docs.kedro.org/>
- Citación: QuantumBlack, McKinsey & Company. (2019). *Kedro: A Python framework for reproducible, maintainable and modular data science code*.

#### 4. LightGBM

- Paper: Ke et al. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*. NIPS.
- URL: <https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree>
- GitHub: <https://github.com/microsoft/LightGBM>

#### 5. XGBoost

- Paper: Chen & Guestrin. (2016). *XGBoost: A Scalable Tree Boosting System*. KDD '16.
- URL: <https://doi.org/10.1145/2939672.2939785>
- GitHub: <https://github.com/dmlc/xgboost>

#### 6. Scikit-learn

- Paper: Pedregosa et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR 12, pp. 2825-2830.
- URL: <https://jmlr.org/papers/v12/pedregosa11a.html>
- Documentación: <https://scikit-learn.org/>

## 13.3 Metodologías

### 7. CRISP-DM

- Chapman et al. (2000). *CRISP-DM 1.0: Step-by-step data mining guide.*
- URL: <https://www.the-modeling-agency.com/crisp-dm.pdf>

### 8. MLOps

- Sculley et al. (2015). *Hidden Technical Debt in Machine Learning Systems.* NIPS.
- URL: <https://papers.nips.cc/paper/2015/file/86df7dcfd896caf2674f757a2463eba-Paper.pdf>

### 9. Docker Best Practices

- Docker Inc. (2023). *Docker Development Best Practices.*
- URL: <https://docs.docker.com/develop/dev-best-practices/>

## 13.4 Recursos Adicionales

### 1. Kedro Tutorials Seguidos

- Official Kedro Tutorial: [https://docs.kedro.org/en/stable/tutorial/spaceflights\\_tutorial.html](https://docs.kedro.org/en/stable/tutorial/spaceflights_tutorial.html)
- Real-world ML project with Kedro: <https://kedro.org/blog/>

### 2. ML Best Practices

- Google. (2023). *Rules of Machine Learning: Best Practices for ML Engineering.*
- URL: <https://developers.google.com/machine-learning/guides/rules-of-ml>

## 13.5 Herramientas

### 1. DVC (Data Version Control)

- Documentación: <https://dvc.org/doc>
- GitHub: <https://github.com/iterative/dvc>

### 2. Docker

- Documentación: <https://docs.docker.com/>
- Whitepaper interno: `docs/referencias/docker_SUMMARY.md`
- Resumen técnico: `docs/referencias/docker_SUMMARY.md`
- Beneficios clave: Reproducibilidad, aislamiento de dependencias, onboarding acelerado, despliegues previsibles

### 3. Jupyter

- Project Jupyter: <https://jupyter.org/>
- Notebook best practices: <https://jupyter-notebook.readthedocs.io/>

## 13.6 Código del Proyecto

### 1. Repositorio GitHub

- URL: [https://github.com/HecAguilaV/ML\\_Analisis\\_Ecosistema\\_Dev](https://github.com/HecAguilaV/ML_Analisis_Ecosistema_Dev)
- Licencia: MIT License
- Branch principal: `main`

### 2. Notebooks de Análisis

- `02_analisis_de_resultados.ipynb` : Evaluación completa de modelos
- `03_ecosystem_analysis.ipynb` : Análisis del panorama tecnológico
- Ubicación: `notebooks/`

---

## Metadata del Documento

- **Versión:** Actualizada con datos 2023-2025 e IA
- **Fecha:** Noviembre de 2025

- **Autor:** Héctor Aguila V.
  - **Email:** [he.aguila@duocuc.cl](mailto:he.aguila@duocuc.cl)
  - **GitHub:** [@HecAguilaV](https://github.com/HecAguilaV)
  - **Institución:** DuocUC - Ingeniería en Informática
  - **Proyecto:** Análisis Predictivo del Mercado Tech con Perspectiva Regional (Chile)
-