MÓDULO 05: INGENIERÍA DE DATOS Y DVC

Versionado de Datos, DAGs y Reproducibilidad

Guía MLOps v5.0: Senior Edition | DuqueOM | Noviembre 2025

MÓDULO 05: Ingeniería de Datos y DVC

El Arte de Versionar lo que Git No Puede

"Si no puedo recrear tus datos, no puedo reproducir tu modelo."

Duración	Teoría	Práctica
5-6 horas	30%	70%

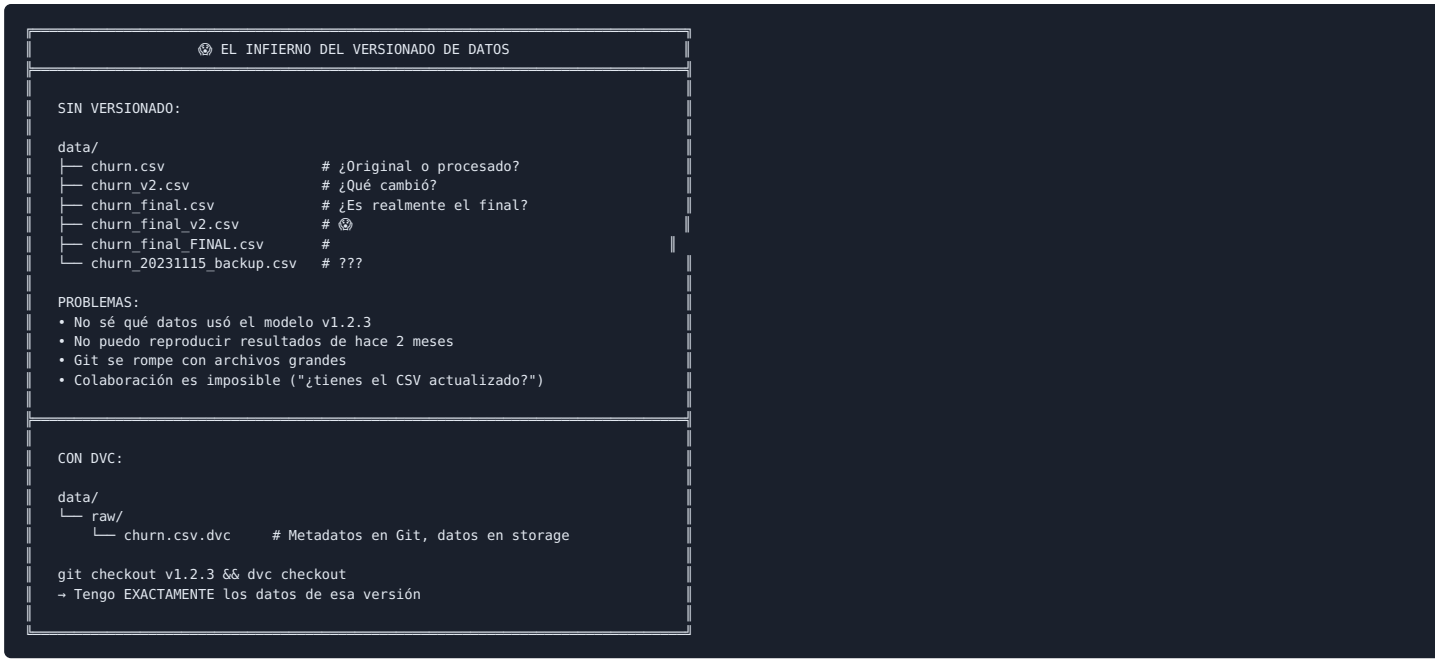
ADR de Inicio: ¿Cuándo (NO) Usar DVC?

```
ADR-006: Criterios para Usar DVC

 USA DVC SI:
• Datos > 100MB que no caben cómodamente en Git
• Necesitas reproducibilidad exacta de datasets
• Equipo colabora en el mismo pipeline de datos
• Quieres DAGs declarativos para pipelines
• Datos son batch (no streaming)

× NO USES DVC SI:
• Datos < 50MB y no cambian frecuentemente → Git LFS o Git directo
• Datos son streaming (Kafka, Kinesis) → No aplica versionado batch
• Ya tienes Data Lake con Delta Lake/Iceberg → Usar versionado nativo
• Solo 1 persona trabaja en el proyecto → Puede ser overkill
• Pipeline ya está en Airflow/Prefect → Evitar duplicación

DECISIÓN PARA BANKCHURN:
Usar DVC porque: datos ~50MB con potencial de crecer, equipo colabora,
queremos reproducibilidad completa, y el pipeline es batch.
```

Lo Que Lograrás en Este Módulo

1. **Entender** el problema del versionado de datos en ML
2. **Configurar** DVC con remote storage
3. **Crear** pipelines reproducibles con `dvc.yaml`
4. **Diseñar** DAGs para proyectos complejos

5.1 El Problema: Git No Escala para Datos



Comparativa de Soluciones

Solución	Tamaño Máx	Versionado	Pipelines	Costo	Complejidad
Git directo	~10MB		×	Gratis	Baja
Git LFS	~2GB		×	$$$	Baja
DVC	Ilimitado			Storage	Media
Delta Lake	Ilimitado		×	Spark	Alta
LakeFS	Ilimitado		×	Server	Alta

5.2 Configuración Inicial de DVC

Instalación

```
# Con pip
pip install dvc

# Con extras para storage
pip install "dvc[s3]"      # Amazon S3
pip install "dvc[gs]"      # Google Cloud Storage
pip install "dvc[azure]"   # Azure Blob Storage
pip install "dvc[gdrive]"  # Google Drive (para proyectos personales)
```

Inicialización

```
# En un repo Git existente
cd bankchurn-predictor
dvc init

# Esto crea:
# .dvc/            - Directorio de configuración
# .dvc/.gitignore
# .dvc/config
# .dvcignore       - Qué ignorar (como .gitignore)
```

Configurar Remote Storage

```
# ═══════════════════════════════════════════════════════════
# OPCIÓN 1: Local (para desarrollo)
# ═══════════════════════════════════════════════════════════
dvc remote add -d localremote /path/to/dvc-storage
# -d = default remote

# ═══════════════════════════════════════════════════════════
# OPCIÓN 2: Amazon S3
# ═══════════════════════════════════════════════════════════
dvc remote add -d s3remote s3://my-bucket/dvc-storage
dvc remote modify s3remote region us-east-1
# Credenciales: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY en env

# ═══════════════════════════════════════════════════════════
# OPCIÓN 3: Google Cloud Storage
# ═══════════════════════════════════════════════════════════
dvc remote add -d gcsremote gs://my-bucket/dvc-storage
# Credenciales: GOOGLE_APPLICATION_CREDENTIALS en env

# ═══════════════════════════════════════════════════════════
# OPCIÓN 4: Google Drive (Gratis, bueno para proyectos personales)
# ═══════════════════════════════════════════════════════════
dvc remote add -d gdriveremote gdrive://folder-id
# La primera vez pedirá autenticación OAuth

# ═══════════════════════════════════════════════════════════
# Ver configuración
# ═══════════════════════════════════════════════════════════
cat .dvc/config
```

Estructura de Directorios Recomendada

```
bankchurn-predictor/
├── data/
│   ├── raw/                    # Datos originales (DVC tracked)
│   │   ├── .gitkeep
│   │   └── churn.csv           # → churn.csv.dvc en Git
│   ├── processed/              # Datos procesados (output de pipeline)
│   │   └── .gitkeep
│   └── external/               # Datos de terceros
│       └── .gitkeep
├── models/                     # Modelos entrenados (DVC tracked)
│   └── .gitkeep
├── .dvc/
│   └── config
├── .dvcignore
└── dvc.yaml                    # Pipeline definition
```

5.3 Versionado Básico de Archivos

Añadir Datos a DVC

```
# Añadir archivo
dvc add data/raw/churn.csv

# Esto crea:
# data/raw/churn.csv.dvc   - Metadatos (hash, size)
# data/raw/.gitignore      - Ignora el CSV en Git

# Ver contenido del .dvc
cat data/raw/churn.csv.dvc
```

```
# data/raw/churn.csv.dvc
outs:
- md5: abc123def456...
  size: 52428800
  hash: md5
  path: churn.csv
```

Flujo de Trabajo

```
# 1. Modificar datos
# ... (actualizar churn.csv con nuevos registros)

# 2. Actualizar tracking
dvc add data/raw/churn.csv

# 3. Commit ambos cambios
git add data/raw/churn.csv.dvc data/raw/.gitignore
git commit -m "data(raw): update churn dataset with Q4 2024 data"

# 4. Push datos a remote
dvc push

# 5. Push código a Git
git push
```

Recuperar Datos de Versión Anterior

```
# Ver versiones del archivo
git log data/raw/churn.csv.dvc

# Checkout versión específica
git checkout v1.0.0 -- data/raw/churn.csv.dvc
dvc checkout data/raw/churn.csv

# O más simple: checkout todo
git checkout v1.0.0
dvc checkout
# → Ahora tienes código Y datos de v1.0.0
```

5.4 Pipelines con dvc.yaml (El Poder Real)

¿Por Qué Pipelines?

```
╔═══════════════════════════════════════════════════════════════╗
║                    PIPELINES DVC: REPRODUCIBILIDAD             ║
╠═══════════════════════════════════════════════════════════════╣
║                                                               ║
║  SIN PIPELINE:                                                ║
║  "Para reproducir, ejecuta preprocess.py, luego train.py, luego..."  ║
║  "Ah, pero primero asegúrate de tener los datos correctos..."  ║
║  "Y usa los mismos hiperparámetros que están en... algún lugar..."  ║
║                                                               ║
║  CON PIPELINE DVC:                                            ║
║  $ dvc repro                                                  ║
║  → Ejecuta TODO automáticamente, en orden correcto,           ║
║    saltando stages que no cambiaron.                          ║
║                                                               ║
╚═══════════════════════════════════════════════════════════════╝
```

dvc.yaml Completo para BankChurn

```
# dvc.yaml
stages:
  # ═══════════════════════════════════════════════════════════
  # STAGE 1: Preparación de Datos
  # ═══════════════════════════════════════════════════════════
  prepare:
    cmd: python src/bankchurn/data/prepare.py
    deps:
      - src/bankchurn/data/prepare.py
      - data/raw/churn.csv
      - configs/config.yaml
    params:
      - prepare.test_size
      - prepare.random_state
    outs:
      - data/processed/train.csv
      - data/processed/test.csv

  # ═══════════════════════════════════════════════════════════
  # STAGE 2: Feature Engineering
  # ═══════════════════════════════════════════════════════════
  featurize:
    cmd: python src/bankchurn/features/build.py
    deps:
      - src/bankchurn/features/build.py
      - data/processed/train.csv
      - data/processed/test.csv
      - configs/config.yaml
    params:
      - features.numerical
      - features.categorical
    outs:
      - data/processed/train_features.pkl
      - data/processed/test_features.pkl

  # ═══════════════════════════════════════════════════════════
  # STAGE 3: Entrenamiento
  # ═══════════════════════════════════════════════════════════
  train:
    cmd: python src/bankchurn/training.py
    deps:
      - src/bankchurn/training.py
      - data/processed/train_features.pkl
      - configs/config.yaml
    params:
      - train.n_estimators
      - train.max_depth
      - train.random_state
    outs:
      - models/pipeline.pkl
    metrics:
      - metrics/train_metrics.json:
          cache: false

  # ═══════════════════════════════════════════════════════════
  # STAGE 4: Evaluación
  # ═══════════════════════════════════════════════════════════
  evaluate:
    cmd: python src/bankchurn/evaluate.py
    deps:
      - src/bankchurn/evaluate.py
      - models/pipeline.pkl
      - data/processed/test_features.pkl
    metrics:
      - metrics/eval_metrics.json:
          cache: false
    plots:
      - metrics/roc_curve.json:
          x: fpr
          y: tpr
      - metrics/confusion_matrix.json:
          template: confusion
          x: predicted
          y: actual
```

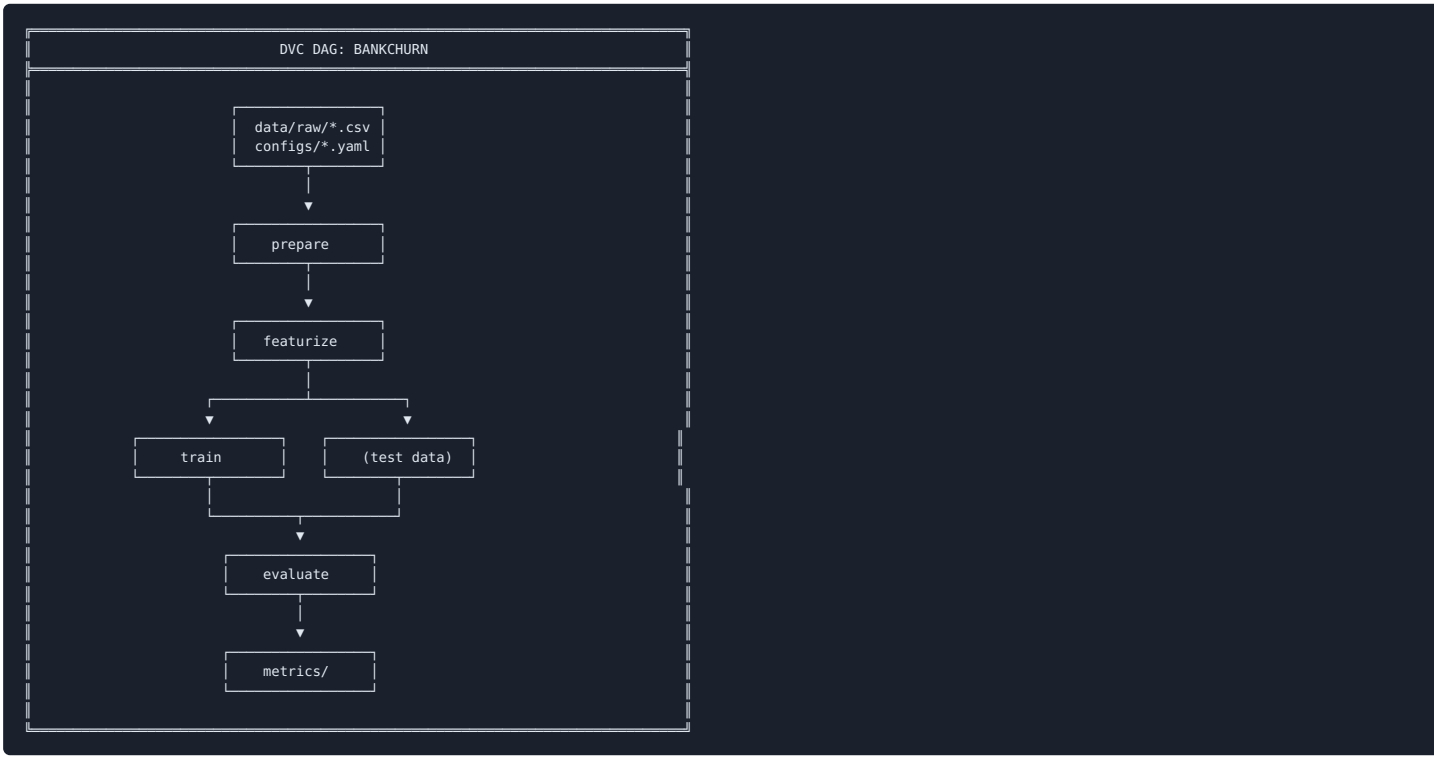
params.yaml (Configuración del Pipeline)

```yaml
# params.yaml
prepare:
  test_size: 0.2
  random_state: 42

features:
  numerical:
    - CreditScore
    - Age
    - Tenure
    - Balance
    - NumOfProducts
    - EstimatedSalary
  categorical:
    - Geography
    - Gender

train:
  n_estimators: 100
  max_depth: 10
  random_state: 42
```

Comandos de Pipeline

```bash
# ═══════════════════════════════════════════════════════════
# REPRODUCIR PIPELINE
# ═══════════════════════════════════════════════════════════

# Ejecutar todo el pipeline
dvc repro

# Ejecutar stage específico (y sus dependencias)
dvc repro train

# Forzar re-ejecución (aunque no haya cambios)
dvc repro --force

# Ver qué se ejecutaría sin ejecutar
dvc repro --dry

# ═══════════════════════════════════════════════════════════
# VISUALIZAR PIPELINE
# ═══════════════════════════════════════════════════════════

# Ver DAG en terminal
dvc dag

# Generar imagen del DAG
dvc dag --dot | dot -Tpng -o pipeline.png

# Ver dependencias de un stage
dvc dag --outs train
```

Visualización del DAG



5.5 Métricas y Experimentos

Tracking de Métricas

```
# Ver métricas actuales
dvc metrics show

# Comparar con otra rama/commit
dvc metrics diff HEAD~1

# Output ejemplo:
# Path                    Metric    HEAD     HEAD~1   Change
# metrics/eval_metrics.json  auc_roc   0.8721   0.8534   0.0187
# metrics/eval_metrics.json  f1        0.7234   0.7012   0.0222
```

Experimentos con DVC

```
# ═══════════════════════════════════════════════════════════
# EJECUTAR EXPERIMENTOS
# ═══════════════════════════════════════════════════════════

# Experimento con cambio de parámetro
dvc exp run --set-param train.n_estimators=200

# Múltiples experimentos en paralelo
dvc exp run --queue --set-param train.n_estimators=100
dvc exp run --queue --set-param train.n_estimators=200
dvc exp run --queue --set-param train.n_estimators=300
dvc exp run --run-all --parallel 3

# ═══════════════════════════════════════════════════════════
# COMPARAR EXPERIMENTOS
# ═══════════════════════════════════════════════════════════

# Ver todos los experimentos
dvc exp show

# Output:
#
# ┌──────────────┬──────────┬──────────┬──────────────┐
# │ Experiment   │ auc_roc  │ f1       │ n_estimators │
# ├──────────────┼──────────┼──────────┼──────────────┤
# │ main         │ 0.8721   │ 0.7234   │ 100          │
# │ exp-abc123   │ 0.8856   │ 0.7421   │ 200          │
# │ exp-def456   │ 0.8812   │ 0.7356   │ 300          │
# └──────────────┴──────────┴──────────┴──────────────┘

# ═══════════════════════════════════════════════════════════
# APLICAR MEJOR EXPERIMENTO
# ═══════════════════════════════════════════════════════════

# Aplicar a workspace
dvc exp apply exp-abc123

# O crear branch
dvc exp branch exp-abc123 feature/best-model
```

5.6 Patrones Avanzados

Multi-Output Stages

```yaml
# dvc.yaml
stages:
  split:
    cmd: python src/split.py
    deps:
      - data/raw/full_dataset.csv
    outs:
      - data/processed/train.csv
      - data/processed/val.csv
      - data/processed/test.csv
```

Stages Condicionales (foreach)

```yaml
# dvc.yaml - Entrenar múltiples modelos
stages:
  train:
    foreach:
      - random_forest
      - xgboost
      - lightgbm
    do:
      cmd: python src/train.py --model ${item}
      deps:
        - src/train.py
        - data/processed/train.csv
      params:
        - train.${item}
      outs:
        - models/${item}.pkl
      metrics:
        - metrics/${item}_metrics.json:
            cache: false
```

Integración con MLflow

```python
# src/bankchurn/training.py
import mlflow
import dvc.api
import yaml

def train():
    # Obtener parámetros de DVC
    params = dvc.api.params_show()

    with mlflow.start_run():
        # Log parámetros
        mlflow.log_params(params["train"])

        # Entrenar...
        model = train_model(params["train"])

        # Log métricas
        metrics = evaluate(model)
        mlflow.log_metrics(metrics)

        # Guardar métricas para DVC también
        with open("metrics/train_metrics.json", "w") as f:
            json.dump(metrics, f)

        # Log modelo
        mlflow.sklearn.log_model(model, "model")
```

5.7 Ejercicio Integrador

Setup Completo de DVC

```bash
# 1. Inicializar DVC
cd bankchurn-predictor
dvc init

# 2. Configurar remote (local para empezar)
mkdir -p ~/dvc-storage
dvc remote add -d localremote ~/dvc-storage

# 3. Crear estructura de datos
mkdir -p data/{raw,processed} models metrics

# 4. Añadir datos raw
# (asumiendo que tienes churn.csv)
cp /path/to/churn.csv data/raw/
dvc add data/raw/churn.csv

# 5. Crear dvc.yaml (copiar del ejemplo anterior)

# 6. Crear params.yaml

# 7. Commit todo
git add .
git commit -m "data(dvc): setup DVC pipeline"

# 8. Ejecutar pipeline
dvc repro

# 9. Push a remote
dvc push
git push
```

Checklist de Verificación

```
CONFIGURACIÓN:
[ ] DVC inicializado
[ ] Remote configurado y funcionando
[ ] Datos raw tracked con DVC

PIPELINE:
[ ] dvc.yaml con stages definidos
[ ] params.yaml con parámetros
[ ] dvc repro ejecuta sin errores

VERSIONADO:
[ ] Puedo hacer git checkout + dvc checkout a versiones anteriores
[ ] dvc push/pull funcionan correctamente
[ ] Métricas se trackean con dvc metrics show
```

5.8 Autoevaluación

Preguntas de Reflexión

1. ¿Por qué DVC usa hashes MD5 en lugar de guardar los archivos?
2. ¿Qué pasa si cambio `params.yaml` pero no el código?
3. ¿Cuándo DVC salta un stage sin ejecutarlo?
4. ¿Cómo integrarías DVC con GitHub Actions para CI?

Siguiente Paso

Con datos versionados, es hora de construir **pipelines de Sklearn avanzados**.

Ir a Módulo 06: Pipelines Sklearn Avanzados →

Módulo 05 completado. Tus datos ahora tienen historial como tu código.