Simulacro de Entrevista Lead/Senior ML Engineer

Portafolio MLOps — 100+ Preguntas Técnicas y Conceptuales

Autor del Portafolio: Daniel Duque (DuqueOM)
Versión: 1.0
Fecha: Noviembre 2025

Índice

---

1. Fundamentos de Machine Learning (Preguntas 1-15)

Pregunta 1: Data Leakage

¿Qué es el data leakage y cómo lo preveniste?

Respuesta:

El data leakage ocurre cuando información del test set o target filtra al training.

Prevención en BankChurn (`training.py:278-293`):

```
        # CRITICAL: Split BEFORE fitting preprocessor
        X_train, X_test, y_train, y_test = train_test_split(X, y, ...)
        self.preprocessor_ = self.build_preprocessor(X_train)  # Solo training
        X_train = self.preprocessor_.fit_transform(X_train)
        X_test = self.preprocessor_.transform(X_test)  # Solo transform
```

En CarVision (`config.yaml`):

```
        drop_columns: ["price_per_mile", "price_category"]  # Dependen del target
```

Justificación: El preprocesador (StandardScaler, OneHotEncoder) debe ajustarse **exclusivamente** con datos de training.

Pregunta 2: Class Imbalance

¿Cómo manejaste el desbalance 80/20 en churn?

Respuesta:

Implementé múltiples estrategias en `ResampleClassifier` (`models.py`):

```
class ResampleClassifier(BaseEstimator, ClassifierMixin):
    # strategy: {"none", "oversample", "undersample", "class_weight"}
    def _apply_resampling(self, X, y):
        if self.strategy == "oversample":
            return SMOTE(random_state=self.random_state).fit_resample(X, y)
        elif self.strategy == "undersample":
            return RandomUnderSampler().fit_resample(X, y)
```

Estrategias adicionales: - `LogisticRegression(class_weight="balanced")` - `RandomForestClassifier(class_weight="balanced_subsample")` - **F1-Score** como métrica primaria (equilibra precision-recall)

Pregunta 3: Ensemble VotingClassifier

Explica el VotingClassifier con pesos [0.4, 0.6].

Respuesta:

```
ensemble = VotingClassifier(
    estimators=[("lr", lr), ("rf", rf)],
    voting="soft",      # Promedia probabilidades
    weights=[0.4, 0.6]  # RF tiene mayor peso
)
```

Razones: 1. **Soft voting**: Aprovecha confianza del modelo (probabilidades) 2. **RF (0.6)**: Mejor AUC individual, captura nc linealidades 3. **LR (0.4)**: Regularización, interpretabilidad, buen baseline

Complementariedad: LR asume linealidad, RF captura interacciones → combinación reduce varianza.

Pregunta 4: FeatureEngineer Centralizado

¿Cómo garantizas consistencia entre training e inference?

Respuesta:

Clase `FeatureEngineer` como sklearn transformer (`features.py`):

```
class FeatureEngineer(BaseEstimator, TransformerMixin):
    def transform(self, X):
        X = X.copy()
        if "model_year" in X.columns:
            X["vehicle_age"] = self.current_year - X["model_year"]
        if "model" in X.columns:
            X["brand"] = X["model"].str.split().str[0]
        return X
```

Pipeline integrado:

```
pipe = Pipeline([("features", fe), ("pre", pre), ("model", model)])
joblib.dump(pipe, "model.joblib")  # TODO serializado
```

Beneficio: Single Source of Truth para training, API, dashboard.

Pregunta 5: StratifiedKFold

¿Por qué usaste StratifiedKFold?

Respuesta:

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

Razones: 1. **Preserva proporción**: Con 80/20, KFold simple podría generar folds 90/10 2. **Representatividad**: Cada fol simula distribución de producción 3. **Estabilidad**: Reduce varianza entre folds

---

Pregunta 6: Métricas de Regresión

¿Cuándo priorizas RMSE vs MAE vs MAPE?

Respuesta:

Métrica	Priorizar cuando
RMSE	Errores grandes son costosos (penaliza outliers)
MAE	Robustez a outliers, interpretación directa
MAPE	Comparación entre escalas diferentes
R^2	Benchmarking vs baseline

En CarVision priorizo RMSE: Un error de $10K en auto de $5K es crítico.

Pregunta 7: Imputation Strategies

¿Por qué median para numéricos y "constant" para categóricos?

Respuesta:

```
# Numéricos: robusta a outliers
("imputer", SimpleImputer(strategy="median"))
# Categóricos: categoría explícita permite aprender patrón de missing
("imputer", SimpleImputer(strategy="constant", fill_value="missing"))
```

Justificación: `most_frequent` sesga hacia categoría dominante; "missing" es más informativo.

Pregunta 8: Regularización L2

¿Qué significa C=0.1 en LogisticRegression?

Respuesta:

```
        LogisticRegression(C=0.1, solver="liblinear")
```

C = 1/λ (inverso de regularización L2): - C bajo (0.1) = regularización fuerte → modelo simple, menos overfitting - C alt (10) = regularización débil → más flexibilidad

Selección: Incluido en espacio de búsqueda Optuna `[0.01, 10.0]` log scale.

Pregunta 9: Random Forest Hyperparameters

¿Por qué max_depth=10 y min_samples_leaf=5?

Respuesta:

Parámetro	Valor	Efecto
`max_depth=10`	Límite profundidad	Previene memorización
`min_samples_leaf=5`	Mínimo en hojas	Evita hojas con 1-2 samples
`n_estimators=100`	Árboles	Balance varianza/costo

Validación: ~0.85 AUC con gap train-test < 3%.

Pregunta 10: GradientBoosting vs RandomForest

¿Por qué GB en TelecomAI y RF en BankChurn?

Respuesta:

Aspecto	Random Forest	Gradient Boosting
Enfoque	Bagging (paralelo)	Boosting (secuencial)
Dataset	Grande (10K+)	Pequeño (~2K)
Features	Mix cat/num	Numéricas simples

TelecomAI usa GB con `max_depth=2, lr=0.05`: muchos árboles simples → regularización fuerte.

Pregunta 11: Cross-Validation con Logging

¿Cómo reportas resultados de CV?

Respuesta:

```
        for fold, (train_idx, val_idx) in enumerate(cv.split(X_train, y_train), 1):
            fold_f1 = f1_score(y_fold_val, y_pred, average="weighted")
            cv_scores.append(fold_f1)
            logger.info(f"Fold {fold}/{n_folds}: F1 = {fold_f1:.4f}")

        logger.info(f"CV Mean F1: {np.mean(cv_scores):.4f} (+/- {np.std(cv_scores):.4f})")
```

Métricas reportadas: Media ± desviación estándar para evaluar estabilidad.

Pregunta 12: Threshold Selection

¿Cómo elegiste el threshold de clasificación?

Respuesta:

```
        # prediction.py
        def predict(self, X, threshold=0.5):
            results["prediction"] = (results["probability"] >= threshold).astype(int)
            results["risk_level"] = pd.cut(results["probability"],
                bins=[0, 0.3, 0.7, 1.0], labels=["low", "medium", "high"])
```

Default 0.5, pero configurable vía API. En producción: - Si costo de FN alto → bajar threshold (más recall) - Si costo de F alto → subir threshold (más precision)

Pregunta 13: Feature Importance

¿Cómo calculas y usas feature importance?

Respuesta:

```
        # evaluation.py - error_analysis
        feature_importance: true  # En config

        # RandomForest tiene .feature_importances_
        # Para ensemble, extraemos de sub-modelos
```

Uso: Identificar features dominantes, detectar posibles data leakage, explicabilidad al negocio.

Pregunta 14: Calibration

¿Qué es la calibración de probabilidades?

Respuesta:

Del Model Card:

```
        ## Calibration
        - Platt (sigmoid) sobre el modelo final (cv="prefit")
```

Problema: `predict_proba()` puede dar probabilidades que no reflejan frecuencia real. **Solución**: `CalibratedClassifierCV` ajusta probabilidades para que 0.7 signifique ~70% de positivos reales.

Pregunta 15: Baseline Comparison

¿Cómo comparas tu modelo contra un baseline?

Respuesta:

```
# evaluation.py (CarVision)
dummy = DummyRegressor(strategy="median")
dummy.fit(X_train, y_train)
baseline_metrics = {"rmse": rmse(y_test, dummy.predict(X_test))}
```

Bootstrap para significancia estadística:

```
delta_rmse_mean = modelo - baseline
p_value_two_sided  # Si < 0.05, diferencia significativa
```

---

2. MLOps y Ciclo de Vida (Preguntas 16-30)

Pregunta 16: MLflow Integration

¿Cómo integraste MLflow?

Respuesta:

```
# training.py
if self.config.mlflow.enabled:
    mlflow.set_tracking_uri(self.config.mlflow.tracking_uri)
    mlflow.set_experiment(self.config.mlflow.experiment_name)

with mlflow.start_run():
    mlflow.log_params(self.config.model.dict())
    mlflow.log_metrics(metrics)
```

Config (`config.yaml`):

```
mlflow:
  tracking_uri: "file:./mlruns"  # Local
  experiment_name: "bankchurn-experiment"
```

Pregunta 17: Reproducibilidad con Seeds

¿Cómo garantizas reproducibilidad?

Respuesta:

```
# common_utils/seed.py
def set_seed(seed=None):
    seed = seed or os.getenv("SEED") or 42
    os.environ["PYTHONHASHSEED"] = str(seed)
    random.seed(seed)
    np.random.seed(seed)
    # PyTorch/TensorFlow si instalados
    return seed
```

Uso en todos los proyectos:

```
seed_used = set_seed(args.seed)
logger.info(f"Using seed: {seed_used}")
```

Pregunta 18: Configuration con Pydantic

Explica tu sistema de configuración.

Respuesta:

```
# config.py
class ModelConfig(BaseModel):
    test_size: float = Field(0.2, ge=0.0, le=1.0)
    cv_folds: int = Field(5, ge=2)

class BankChurnConfig(BaseModel):
    model: ModelConfig
    data: DataConfig
    mlflow: MLflowConfig

    @classmethod
    def from_yaml(cls, path):
        with open(path) as f:
            return cls(**yaml.safe_load(f))
```

Beneficios: Validación automática, valores por defecto, type hints, serialización.

Pregunta 19: Model Versioning

¿Cómo versionas modelos?

Respuesta:

1. **Naming**: `models/model_v1.0.0.pkl`
2. **Metadata JSON**: commit_sha, métricas, timestamp
3. **K8s ConfigMap**: `MODEL_VERSION: "v2.0.0"`
4. **Docker tags**: `bankchurn:v1.0.0`
5. **MLflow Registry**: Staging → Production

Pregunta 20: DVC Data Versioning

¿Por qué DVC?

Respuesta:

```
dvc add data/raw/Churn.csv
git add data/raw/Churn.csv.dvc .gitignore
dvc push  # A storage remoto
```

Beneficios: Cada commit Git tiene snapshot exacto de datos, sin almacenarlos en Git.

Pregunta 21: Pipeline como Artefacto Único

¿Por qué empaquetar preprocessor + model?

Respuesta:

```
pipeline = Pipeline([("preprocess", pre), ("clf", clf)])
joblib.dump(pipeline, "model.joblib")
```

Deployment simplificado:

```
pipe = joblib.load("model.joblib")
pred = pipe.predict(raw_df)  # Sin pasos intermedios
```

Pregunta 22: Health Checks

¿Cómo implementaste health checks?

Respuesta:

```
@app.get("/health")
async def health_check():
    return {"status": "healthy" if predictor else "degraded",
            "model_loaded": predictor is not None}
```

K8s probes:

```
livenessProbe:
  httpGet: {path: /health, port: 8000}
  initialDelaySeconds: 30
readinessProbe:
  httpGet: {path: /health, port: 8000}
  initialDelaySeconds: 10
```

Pregunta 23: Model Loading Strategy

¿Cómo cargas el modelo en FastAPI?

Respuesta:

```
@contextlib.asynccontextmanager
async def lifespan(app: FastAPI):
    global predictor
    predictor = load_model_logic()  # Una vez al iniciar
    yield

app = FastAPI(lifespan=lifespan)
```

Beneficio: Carga única, no por request. Fail-fast friendly.

Pregunta 24: Batch Prediction

¿Cómo manejas predicciones batch?

Respuesta:

```python
@app.post("/predict_batch")
async def predict_batch(batch_data: BatchCustomerData):
    if len(batch_data.customers) > 1000:
        raise HTTPException(400, "Max 1000 per batch")

    df = pd.DataFrame([c.dict() for c in batch_data.customers])
    results = predictor.predict(df)
    return {"predictions": results, "processing_time": time}
```

Pregunta 25: Metrics Collection

¿Qué métricas operacionales colectas?

Respuesta:

```python
@app.get("/metrics")
async def get_metrics():
    return {
        "total_predictions": request_count,
        "average_prediction_time_ms": avg_time,
        "model_accuracy": model_metadata.get("test_accuracy")
    }
```

Prometheus-ready: Annotations en K8s para scraping automático.

Pregunta 26: Error Handling en API

¿Cómo manejas errores?

Respuesta:

```python
@app.post("/predict")
async def predict(customer: CustomerData):
    if predictor is None:
        raise HTTPException(503, "Model not available")
    try:
        return predictor.predict(df)
    except Exception as e:
        logger.error(f"Prediction error: {e}")
        raise HTTPException(500, str(e))
```

Códigos apropiados: 503 servicio no disponible, 500 error interno.

Pregunta 27: Logging Strategy

¿Cómo estructuraste el logging?

Respuesta:

```python
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(name)s - %(levelname)s - %(message)s",
    handlers=[FileHandler("app.log"), StreamHandler()]
)
logger = logging.getLogger(__name__)
```

Niveles usados: INFO para operaciones normales, WARNING para fallbacks, ERROR para fallos.

Pregunta 28: API Validation con Pydantic

¿Cómo validas inputs en la API?

Respuesta:

```
class CustomerData(BaseModel):
    CreditScore: int = Field(..., ge=300, le=850)
    Age: int = Field(..., ge=18, le=100)
    Geography: str

    @validator("Geography")
    def validate_geo(cls, v):
        if v not in ["France", "Spain", "Germany"]:
            raise ValueError("Invalid geography")
        return v
```

Pregunta 29: Response Enrichment

¿Qué incluyes en la respuesta de predicción?

Respuesta:

```
class PredictionResponse(BaseModel):
    churn_probability: float
    churn_prediction: int
    risk_level: str         # LOW/MEDIUM/HIGH
    confidence: float       # abs(prob - 0.5) * 2
    model_version: str
    prediction_timestamp: str
```

Risk level: Categorización para decisiones de negocio.

Pregunta 30: Graceful Degradation

¿Qué pasa si el modelo no carga?

Respuesta:

```
async def lifespan(app):
    success = load_model_logic()
    if not success:
        logger.warning("Started without model")
    yield  # App sigue corriendo

# Health check reporta estado degradado
{"status": "degraded", "model_loaded": false}
```

Beneficio: Health checks funcionan, ops pueden diagnosticar.

---

3. BankChurn-Predictor (Preguntas 31-45)

Pregunta 31: Arquitectura Modular

Describe la estructura del proyecto.

Respuesta:

```
src/bankchurn/
├── cli.py        # Entry point CLI
├── config.py     # Pydantic validation
├── models.py     # ResampleClassifier
├── training.py   # ChurnTrainer class
├── evaluation.py # ModelEvaluator class
└── prediction.py # ChurnPredictor class
```

Principio: Single Responsibility - cada módulo una función.

Pregunta 32: CLI Design

¿Cómo diseñaste la CLI?

Respuesta:

```python
# cli.py
parser.add_subparsers(dest="command")
train_parser = subparsers.add_parser("train")
train_parser.add_argument("--config", required=True)

def cli_main(argv=None):
    args = parser.parse_args(argv)
    if args.command == "train":
        return train_command(args)
```

Uso: `python -m bankchurn train --config configs/config.yaml`

Pregunta 33: ResampleClassifier

Explica tu clasificador custom.

Respuesta:

```python
class ResampleClassifier(BaseEstimator, ClassifierMixin):
    def __init__(self, estimator=None, strategy="none"):
        self.estimator = estimator
        self.strategy = strategy

    def fit(self, X, y):
        X_res, y_res = self._apply_resampling(X, y)
        self.estimator_.fit(X_res, y_res)
        return self
```

Implementa interfaz sklearn: `fit`, `predict`, `predict_proba` → compatible con Pipeline.

Pregunta 34: Fairness Evaluation

¿Cómo evalúas sesgo por grupos?

Respuesta:

```
        def compute_fairness_metrics(self, X, y, sensitive_features):
            for feature in sensitive_features:  # ["Geography", "Gender"]
                for group in X[feature].unique():
                    mask = X[feature] == group
                    group_f1 = f1_score(y[mask], y_pred[mask])

            # Disparate Impact
            disparate_impact = min(positive_rates) / max(positive_rates)
```

Threshold: Disparate Impact < 0.8 indica discriminación potencial.

Pregunta 35: Model Card

¿Qué incluyes en tu Model Card?

Respuesta:

```
        # Model Card - BankChurn
        - Model: VotingClassifier (LR + RF)
        - Target: Exited (1=churn)
        ## Intended Use: Priorizar retención, no decisión automática
        ## Limitations: Desbalance 80/20, posible drift temporal
        ## Fairness: Tests por geografía/género, gap recall < 0.3
        ## SLO: Disponibilidad 99.5%, Latencia P95 < 50ms
```

Pregunta 36: Testing Strategy

¿Cómo organizaste los tests?

Respuesta:

```
tests/
├── conftest.py       # Fixtures compartidos
├── test_training.py  # ChurnTrainer
├── test_evaluation.py# ModelEvaluator
├── test_prediction.py# ChurnPredictor
├── test_config.py    # Pydantic validation
└── test_integration.py# E2E workflow
```

Coverage: 77% con pytest-cov.

Pregunta 37: Fixture Pattern

¿Cómo usas fixtures en pytest?

Respuesta:

```python
@pytest.fixture
def config():
    return BankChurnConfig.from_yaml("configs/config.yaml")

@pytest.fixture
def sample_data():
    return pd.DataFrame({
        "CreditScore": np.random.randint(300, 850, 200),
        "Exited": np.random.choice([0, 1], 200, p=[0.8, 0.2])
    })

def test_training(config, sample_data, tmp_path):
    trainer = ChurnTrainer(config)
    # ...
```

Pregunta 38: Integration Test

¿Qué cubre tu test de integración?

Respuesta:

```python
def test_full_training_workflow(config, sample_data, tmp_path):
    trainer = ChurnTrainer(config)
    data = trainer.load_data(data_path)
    X, y = trainer.prepare_features(data)
    model, metrics = trainer.train(X, y)
    trainer.save_model(model_path)

    # Verify artifacts
    assert model_path.exists()
    assert metrics["test_f1"] > 0.5
```

Pregunta 39: Dockerfile Multi-Stage

Explica tu Dockerfile.

Respuesta:

```dockerfile
# Stage 1: Builder
FROM python:3.13-slim AS builder
RUN pip install -r requirements.txt
# Stage 2: Runtime
FROM python:3.13-slim AS runtime
COPY --from=builder /opt/venv /opt/venv
USER appuser  # Non-root
HEALTHCHECK CMD curl -f http://localhost:8000/health
CMD ["uvicorn", "app.fastapi_app:app", "--host", "0.0.0.0"]
```

Beneficios: Imagen final sin build tools, usuario non-root, healthcheck.

Pregunta 40: CORS Configuration

¿Cómo configuraste CORS?

Respuesta:

```python
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Producción: restringir
    allow_methods=["GET", "POST"],
    allow_headers=["*"],
)
```

Nota: `allow_origins=["*"]` solo para desarrollo. Producción especifica dominios.

Pregunta 41-45: [Ver archivo parte 2]

---

4. CarVision-Market-Intelligence (Preguntas 46-60)

Pregunta 46: Pipeline [features, pre, model]

Explica el pipeline de CarVision.

Respuesta:

```
pipe = Pipeline([
    ("features", FeatureEngineer(current_year=2024)),
    ("pre", ColumnTransformer([
        ("num", numeric_pipeline, num_cols),
        ("cat", categorical_pipeline, cat_cols)
    ])),
    ("model", RandomForestRegressor())
])
```

Flujo: Raw DF → Feature Engineering → Preprocessing → Model.

Pregunta 47: Data Filtering

¿Por qué filtras precios entre $1K-$500K?

Respuesta:

```
def clean_data(df, filters):
    df = df[(df["price"] > 1000) & (df["price"] < 500000)]
    df = df[df["model_year"] >= 1990]
    df = df[df["odometer"] < 500000]
```

Razón: < $1K son errores/donaciones, > $500K son coleccionables (mercado diferente).

Pregunta 48: Bootstrap Confidence Intervals

¿Cómo calculas intervalos de confianza?

Respuesta:

```
for _ in range(200):
    idx = rng.choice(n, size=n, replace=True)
    deltas.append(rmse(y[idx], y_model[idx]) - rmse(y[idx], y_base[idx]))

ci_low, ci_high = np.percentile(deltas, [2.5, 97.5])
```

Interpretación: CI95 no incluye 0 → diferencia significativa.

Pregunta 49: Temporal Backtesting

¿Cómo validas con datos temporales?

Respuesta:

```
df_sorted = df.sort_values("model_year")
df_test = df_sorted.tail(int(len(df) * 0.2))  # Más recientes
metrics_temporal = evaluate(model, df_test)
```

Simula producción: Siempre predecimos el "futuro".

Pregunta 50: Streamlit Caching

¿Cómo optimizas el dashboard?

Respuesta:

```
@st.cache_data  # DataFrames (serializable)
def load_data():
    return FeatureEngineer().transform(load_raw_data())

@st.cache_resource  # Modelos (no serializable)
def load_model():
    return joblib.load("model.joblib")
```

Preguntas 51-60: [Ver archivo parte 2]

---

5. TelecomAI (Preguntas 61-70)

Pregunta 61: Simple Feature Set

¿Por qué solo 4 features?

Respuesta:

```
features: [calls, minutes, messages, mb_used]
```

Razón: ~2K muestras, más features = overfitting. AUC 0.84 indica señal suficiente.

Pregunta 62: GradientBoosting Shallow

¿Por qué max_depth=2?

Respuesta:

```
        model:
          name: gradient_boosting
          params: {n_estimators: 200, max_depth: 2, learning_rate: 0.05}
```

Muchos árboles simples: Regularización fuerte, reduce overfitting en dataset pequeño.

Pregunta 63-70: [Ver archivo parte 2]

---

6. Arquitectura y Diseño (Preguntas 71-80)

Pregunta 71: Design Patterns

¿Qué patrones aplicaste?

Respuesta:

1. **Strategy**: `ResampleClassifier(strategy="oversample")`
2. **Factory**: `build_model(cfg)` crea diferentes clasificadores
3. **Template Method**: `ChurnTrainer.train()` con pasos customizables
4. **Dependency Injection**: `ChurnTrainer(config)`

Pregunta 72: SOLID Principles

¿Cómo aplicaste SOLID?

Respuesta:

- **S**: Un módulo, una responsabilidad (training.py solo entrena)
- **O**: ResampleClassifier abierto a nuevas estrategias
- **L**: Hereda de sklearn, sustituible donde se espere classifier
- **I**: Interfaces pequeñas (Predictor, Evaluator, Trainer)
- **D**: Depende de config, no de valores hardcodeados

Preguntas 73-80: [Ver archivo parte 2]

---

7. CI/CD y DevOps (Preguntas 81-90)

Pregunta 81: Unified CI Pipeline

Explica tu workflow unificado.

Respuesta:

```yaml
# ci-mlops.yml
jobs:
  tests:           # pytest + coverage por proyecto
  quality-gates:   # black, flake8, mypy
  security:        # bandit, gitleaks, pip-audit
  docker:          # build + trivy scan
  integration-test: # docker-compose E2E
```

Matrix: Python 3.11/3.12 × 3 proyectos.

Pregunta 82: Security Scanning

¿Qué herramientas de seguridad usas?

Respuesta:

- **Gitleaks**: Detecta secrets en código
- **Bandit**: Análisis estático Python
- **Trivy**: Vulnerabilidades en containers
- **pip-audit**: Dependencias con CVEs

Preguntas 83-90: [Ver archivo parte 2]

---

8. Infraestructura (Preguntas 91-100)

Pregunta 91: Kubernetes Deployment

Explica tu deployment de K8s.

Respuesta:

```yaml
apiVersion: apps/v1
kind: Deployment
spec:
  replicas: 3
  strategy: {type: RollingUpdate}
  template:
    spec:
      containers:
      - name: bankchurn-api
        resources:
          requests: {memory: "512Mi", cpu: "250m"}
          limits: {memory: "1Gi", cpu: "1000m"}
        livenessProbe: ...
        readinessProbe: ...
```

Pregunta 92: HorizontalPodAutoscaler

¿Cómo configuras autoscaling?

Respuesta:

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
spec:
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource: {name: cpu, target: {averageUtilization: 70}}
```

Preguntas 93-100: [Ver archivo parte 2]

---

9. Ética y Fairness (Preguntas 101-105)

Pregunta 101: Responsible AI

¿Cómo abordas responsabilidad en ML?

Respuesta:

1. **Model Cards**: Documentan limitaciones y riesgos
2. **Fairness metrics**: Disparate Impact por grupo
3. **Transparencia**: Logs de predicciones, versiones
4. **Human-in-loop**: Modelo como apoyo, no decisión final

Pregunta 102-105: [Ver archivo parte 2]

---

10. Liderazgo (Preguntas 106-115)

Pregunta 106: Technical Decision Making

¿Cómo decides entre RF y GB?

Respuesta:

Factores: Tamaño dataset, tipo de features, requerimientos de latencia, interpretabilidad. **Proceso**: Experimento co baseline → comparar métricas → considerar trade-offs.

Preguntas 107-115: [Ver archivo Parte 2]

[Continúa en SIMULACRO_ENTREVISTA_PARTE2.md]