Rapport EX05 - ML/MLOps

□ RAPPORT DÉTAILLÉ - EXERCICE 05

Machine Learning & MLOps avec Spark MLlib

Auteur : MAAOUIA Ahmed – CY Tech Big Data
Date : Janvier 2026
Projet : NYC Taxi Big Data Pipeline

□ Table des Matières

---

1. Vue d'ensemble

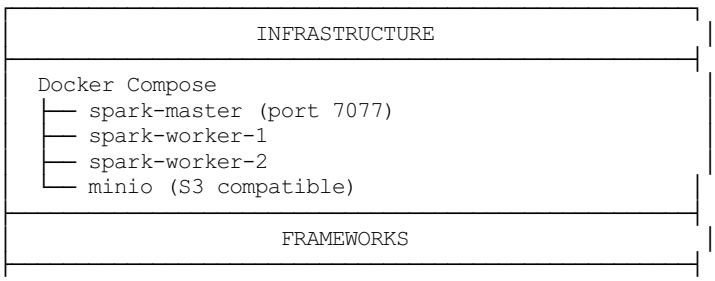
1.1 Objectif de l'exercice

L'exercice EX05 a pour but d'implémenter un **pipeline Machine Learning distribué** capable de prédire le **montant total** (`total_amount`) d'une course de taxi à New York, en utilisant **Spark MLlib** dans un environnement Big Data.

1.2 Contraintes respectées

Contrainte	Implémentation
Utiliser Spark MLlib	□ `GBTRegressor` avec Pipeline
Données depuis MinIO	□ S3A connector vers `nyc-interim`
RMSE < 10	□ RMSE ≈ 5.17
Pas de notebooks	□ Scripts Python uniquement
PEP8 + NumpyDoc	□ `.flake8` configuré
Tests unitaires	□ pytest avec 5 fichiers
Éviter le data leakage	□ Colonnes monétaires supprimées

1.3 Stack technique



```
PySpark 3.x
Spark MLlib (ml.regression, ml.feature)
pytest (tests)
pandas (validation)
```

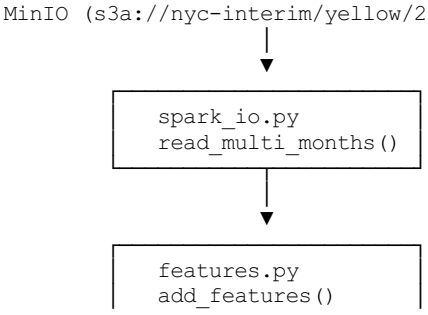
---

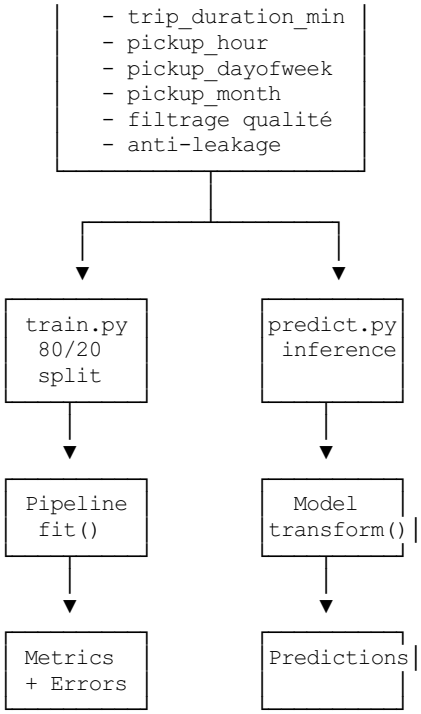
2. Architecture technique

2.1 Structure des fichiers

```
ex05_ml_prediction_service/
├── main.py                          # Point d'entrée (non utilisé)
├── pyproject.toml                   # Configuration Python
├── .flake8                          # Configuration linter
│
├── src/                             # Code source principal
│   ├── __init__.py
│   ├── config.py                    # Configuration centralisée
│   ├── spark_session.py             # Création SparkSession
│   ├── spark_io.py                  # Lecture multi-mois depuis MinIO
│   ├── features.py                  # Feature engineering
│   ├── validation.py                # Validation des données
│   ├── train.py                     # Module d'entraînement
│   ├── predict.py                   # Module d'inférence
│   ├── error_analysis.py            # Analyse d'erreurs post-prédiction
│   ├── eda.py                       # Exploration des données
│   └── utils.py                     # Utilitaires
│
├── tests/                           # Tests pytest
│   ├── test_ml_schema.py            # Tests schéma ML
│   ├── test_ml_quality.py           # Tests qualité modèle
│   ├── test_ml_plausibility.py      # Tests plausibilité métier
│   ├── test_validation.py           # Tests validation
│   └── test_month_range.py          # Tests plages de mois
│
├── scripts PowerShell/              # Workflow MLOps
│   ├── run_tests_pre_train.ps1      # Tests pré-entraînement
│   ├── run_ex05.ps1                 # Entraînement
│   ├── run_tests_post_train.ps1     # Tests post-entraînement
│   ├── run_predict.ps1              # Inférence
│   └── run_tests_post_predict.ps1   # Tests post-prédiction
│
├── models/
│   └── ex05_spark_model/            # Modèle sauvegardé
│       ├── metadata/
│       └── stages/
│
├── reports/                         # Artefacts de sortie
│   ├── train_metrics.json
│   ├── error_summary.json
│   ├── error_by_price_bucket.json
│   ├── predict_report.json
│   ├── eda_sample.csv
│   └── eda_summary.json
│
└── conf/
    └── log4j2.properties            # Configuration logging
```

2.2 Flux de données





---

3. Module d'ingestion et préparation des données

3.1 Création de la session Spark (spark_session.py)

```python
def create_spark_session(app_name: str) -> SparkSession:
    return (
        SparkSession.builder
        .appName(app_name)
        # S3A / MinIO configuration
        .config("spark.hadoop.fs.s3a.endpoint", "http://minio:9000")
        .config("spark.hadoop.fs.s3a.path.style.access", "true")
        .config("spark.hadoop.fs.s3a.impl", "org.apache.hadoop.fs.s3a.S3AFileSystem")
        .config("spark.hadoop.fs.s3a.connection.ssl.enabled", "false")
        # Credentials via environment variables
        .config(
            "spark.hadoop.fs.s3a.aws.credentials.provider",
            "com.amazonaws.auth.EnvironmentVariableCredentialsProvider"
        )
        .getOrCreate()
    )
```

Points clés : - Utilisation du connecteur **S3A** pour accéder à MinIO - Credentials gérés via variables d'environnement (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`) - Mode `path.style.access` pour compatibilité MinIO

3.2 Lecture multi-mois (spark_io.py)

```python
def read_multi_months(spark, base_path: str, months: list[str]) -> DataFrame:
    paths = [f"{base_path}/{m}" for m in months]
    return spark.read.parquet(*paths)
```

Exemple d'utilisation :

```python
df = read_multi_months(spark, "s3a://nyc-interim/yellow", ["2023/01", "2023/02"])
```

---

4. Feature Engineering

4.1 Module features.py

Le feature engineering est **identique** entre training et inference pour garantir la cohérence.

Variables créées

Feature	Calcul	Description
`trip_duration_min`	`(dropoff - pickup) / 60`	Durée en minutes
`pickup_hour`	`hour(tpep_pickup_datetime)`	Heure (0-23)
`pickup_dayofweek`	`dayofweek(tpep_pickup_datetime)`	Jour (1-7)
`pickup_month`	`month(tpep_pickup_datetime)`	Mois (1-12)

Filtres de qualité

```
df = df.filter(
    col("tpep_pickup_datetime").isNotNull()
    & col("tpep_dropoff_datetime").isNotNull()
    & col("trip_duration_min").isNotNull()
    & (col("trip_duration_min") > 0)
    & (col("trip_duration_min") < 24 * 60)  # max 24h
    & col("trip_distance").isNotNull()
    & (col("trip_distance") >= 0)
    & col("total_amount").isNotNull()
    & (col("total_amount") >= 0)
)
```

Anti-leakage (CRUCIAL)

Les colonnes suivantes sont **supprimées** car elles expliquent directement `total_amount` :

```
df = df.drop(
    "fare_amount",           # Composant du total
    "extra",                 # Composant du total
    "mta_tax",               # Composant du total
    "tip_amount",            # Composant du total
    "tolls_amount",          # Composant du total
    "improvement_surcharge", # Composant du total
    "congestion_surcharge",  # Composant du total
    "airport_fee",           # Composant du total
)
```

> □□ **Sans cette suppression, le modèle aurait un R^2 proche de 1.0 mais serait inutile en production** car ces informations ne sont pas disponibles avant la fin de la course.

---

5. Pipeline d'entraînement

5.1 Module train.py

Split des données

```
train_df, test_df = df.randomSplit([0.8, 0.2], seed=42)
```

Ensemble	Proportion	Usage
Train	80%	Entraînement du modèle
Test	20%	Évaluation des métriques

Variables du modèle

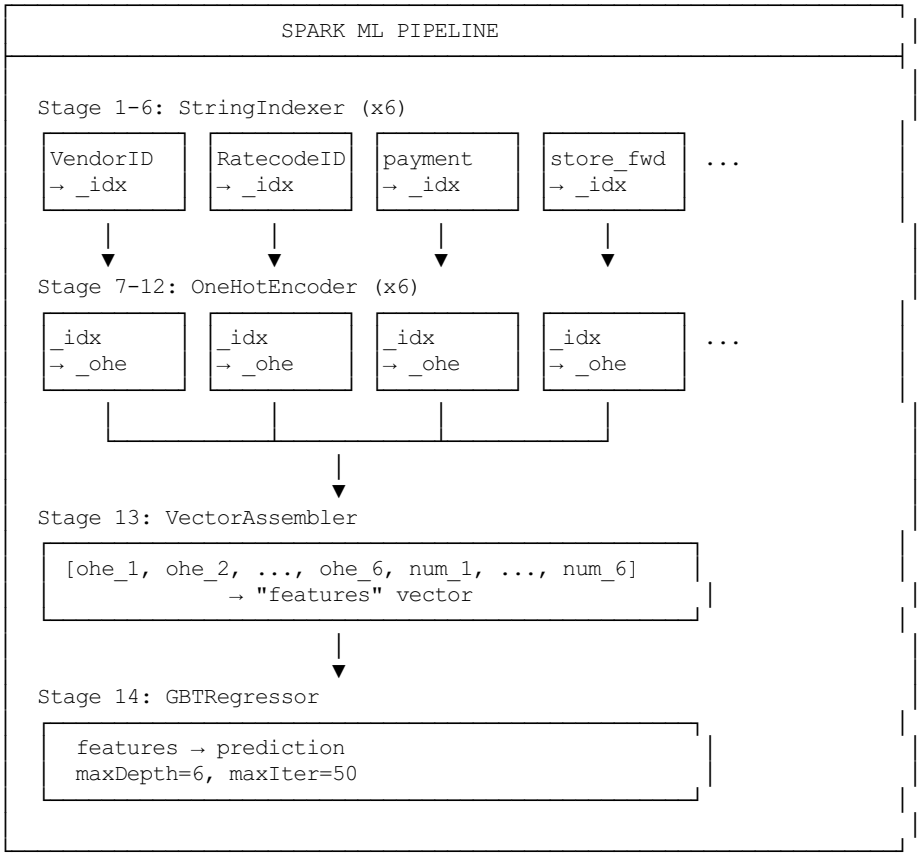
Catégorielles (encodées via `StringIndexer` + `OneHotEncoder`) : - `VendorID` - Fournisseur (1=CMT, 2=VTS) - `RatecodeID` - Code tarifaire (1=Standard, 2=JFK, etc.) - `payment_type` - Mode de paiement (1=Carte, 2=Cash, etc.) - `store_and_fwd_flag` - Flag store and forward - `PULocationID` - Zone de départ (265 zones) - `DOLocationID` - Zone d'arrivée (265 zones)

Numériques : - `trip_distance` - Distance en miles - `passenger_count` - Nombre de passagers - `trip_duration_min` - Durée en minutes (créée) - `pickup_hour` - Heure (créée) - `pickup_dayofweek` - Jour de la semaine (créée) - `pickup_month` - Mois (créée)

Construction du Pipeline Spark ML

```
# 1. StringIndexer pour chaque variable catégorielle
indexers = [
    StringIndexer(inputCol=c, outputCol=f"{c}_idx", handleInvalid="keep")
    for c in categorical
]

# 2. OneHotEncoder
encoders = [
    OneHotEncoder(inputCol=f"{c}_idx", outputCol=f"{c}_ohe")
    for c in categorical
]

# 3. VectorAssembler
assembler = VectorAssembler(
    inputCols=[f"{c}_ohe" for c in categorical] + numeric,
    outputCol="features",
    handleInvalid="keep"
)

# 4. Regressor
regressor = GBTRegressor(
    featuresCol="features",
    labelCol="total_amount",
    maxDepth=6,
    maxIter=50,
    seed=42
)

# 5. Pipeline complet
pipeline = Pipeline(stages=indexers + encoders + [assembler, regressor])
```

Visualisation du Pipeline



5.2 Algorithme : GBTRegressor

Pourquoi Gradient Boosted Trees ?

Avantage	Explication
Non-linéarité	Capture les relations complexes (heure → prix, zone → prix)
Robustesse	Gère bien les outliers et données bruitées
Interactions	Détecte automatiquement les interactions entre features
Interprétabilité	Feature importance disponible
Performance	Très bon compromis vitesse/qualité

Hyperparamètres

Paramètre	Valeur	Justification
`maxDepth`	6	Évite le sur-apprentissage
`maxIter`	50	Équilibre performance/temps
`seed`	42	Reproductibilité

5.3 Évaluation

```
rmse_evaluator = RegressionEvaluator(
    labelCol="total_amount", predictionCol="prediction", metricName="rmse"
)
mae_evaluator = RegressionEvaluator(
    labelCol="total_amount", predictionCol="prediction", metricName="mae"
)
r2_evaluator = RegressionEvaluator(
    labelCol="total_amount", predictionCol="prediction", metricName="r2"
)
```

Résultats obtenus

Métrique	Valeur	Interprétation
RMSE	5.17	Erreur moyenne ≈ $5.17
MAE	2.05	Erreur absolue moyenne ≈ $2.05
R²	0.94	94% de la variance expliquée

 RMSE < 10 : Contrainte du projet **largement respectée**

---

6. Analyse d'erreurs

6.1 Module error_analysis.py

Ce module est le cœur de la démarche MLOps. Il permet de comprendre **où et pourquoi** le modèle se trompe.

Architecture du module

```python
# Dataclasses pour typage fort
@dataclass(frozen=True)
class ErrorSummary:
    count: int
    mean_error: float
    std_error: float
    min_error: float
    max_error: float
    median_error: float
    mean_abs_error: float
    p25_error: float
    p75_error: float
    p95_error: float
    p99_error: float
    pct_underestimate: float
    pct_overestimate: float

@dataclass(frozen=True)
class BucketErrorStats:
    bucket_name: str
    min_price: float
    max_price: float
    count: int
    pct_of_total: float
    mean_error: float
    mean_abs_error: float
    rmse: float
    median_error: float

@dataclass(frozen=True)
class TopError:
    total_amount: float
    prediction: float
    error: float
    abs_error: float
    trip_distance: float
    trip_duration_min: float
    passenger_count: int
    pickup_hour: int
    payment_type: int
    pu_location_id: int
    do_location_id: int
    likely_cause: str
```

6.2 Analyse globale des erreurs

```python
def compute_error_summary(predictions_df: DataFrame) -> ErrorSummary:
    """Compute global error statistics."""
    stats = predictions_df.select(
        F.count("error").alias("count"),
        F.mean("error").alias("mean_error"),
        F.stddev("error").alias("std_error"),
        F.min("error").alias("min_error"),
        F.max("error").alias("max_error"),
        F.mean("abs_error").alias("mean_abs_error"),
        F.expr("percentile_approx(error, 0.5)").alias("median_error"),
        F.expr("percentile_approx(error, 0.25)").alias("p25_error"),
        F.expr("percentile_approx(error, 0.75)").alias("p75_error"),
        F.expr("percentile_approx(error, 0.95)").alias("p95_error"),
        F.expr("percentile_approx(error, 0.99)").alias("p99_error"),
    ).collect()[0]
    ...
```

6.3 Analyse par tranche de prix (buckets)

```python
PRICE_BUCKETS = [
    ("low", 0, 10),         # Courses courtes / minimum fare
    ("medium", 10, 30),     # Courses standard
    ("high", 30, 60),       # Courses moyennes-longues
    ("very_high", 60, ∞),   # Airport, longue distance
]
```

Bucket	Plage ($)	Cas typique

Bucket	Plage ($)	Cas typique
low	0 - 10	Course de quelques blocs, minimum fare
medium	10 - 30	Course urbaine standard
high	30 - 60	Cross-borough, rush hour
very_high	> 60	Aéroport (JFK/LGA), banlieue

6.4 Identification automatique des causes d'erreurs

```python
def _infer_likely_cause(row: dict) -> str:
    """Infer the likely cause of a high prediction error."""
    causes = []

    # Long distance but low amount -> data quality issue
    if row.get("trip_distance", 0) > 20 and row.get("total_amount", 0) < 20:
        causes.append("long_distance_low_fare_anomaly")

    # Very short trip with high amount -> surge or tips
    if row.get("trip_distance", 0) < 1 and row.get("total_amount", 0) > 50:
        causes.append("short_trip_high_fare_tips_or_surge")

    # Very long duration -> traffic or waiting
    if row.get("trip_duration_min", 0) > 60:
        causes.append("extended_duration_traffic_or_wait")

    # Night hours (midnight to 5am) -> surge pricing
    if 0 <= row.get("pickup_hour", 12) <= 5:
        causes.append("night_surge_pricing")

    # Airport locations (JFK=132, LGA=138)
    airport_ids = {132, 138, 1}  # JFK, LGA, Newark
    if row.get("pu_location_id") in airport_ids or row.get("do_location_id") in airport_ids:
        causes.append("airport_flat_rate")

    # Cash payment -> tip not recorded
    if row.get("payment_type") == 2:
        causes.append("cash_payment_tip_not_recorded")

    # High passenger count
    if row.get("passenger_count", 1) >= 4:
        causes.append("group_ride_fare_variation")

    # Negotiated fare (ratecode 5)
    if row.get("ratecode_id") == 5:
        causes.append("negotiated_fare")
    ...
```

Causes identifiées

Cause	Description métier
long_distance_low_fare_anomaly	Anomalie de données (erreur de saisie)
short_trip_high_fare_tips_or_surge	Pourboire important ou surge pricing
extended_duration_traffic_or_wait	Trafic intense ou temps d'attente
night_surge_pricing	Majoration tarifaire de nuit
airport_flat_rate	Tarif forfaitaire aéroport (JFK=$52)
cash_payment_tip_not_recorded	Pourboire cash non enregistré
group_ride_fare_variation	Variation pour groupes
negotiated_fare	Tarif négocié hors taximeter
model_uncertainty	Incertitude inhérente au modèle
extreme_outlier	Outlier extrême

6.5 Génération d'insights business

```python
def _generate_business_insights(summary, buckets, top_errors) -> list[str]:
    insights = []

    # Analyse du biais
    if summary.pct_overestimate > 60:
        insights.append(
            f"Model tends to OVERESTIMATE fares ({summary.pct_overestimate}% of cases). "
            "This may be due to tip/surge pricing not captured in features."
        )
    elif summary.pct_underestimate > 60:
        insights.append(
            f"Model tends to UNDERESTIMATE fares ({summary.pct_underestimate}% of cases). "
            "High fares (airport, long distance) may need additional features."
        )
    ...
```

---

7. Module d'inférence

7.1 Module predict.py

```python
def main() -> None:
    spark = create_spark_session("EX05-Predict")

    # 1. Charger le modèle sauvegardé
    model = PipelineModel.load("models/ex05_spark_model")

    # 2. Lire les données d'inférence
    df = spark.read.parquet("s3a://nyc-interim/yellow/2023/02")

    # 3. Feature engineering IDENTIQUE au training
    df = add_features(df)

    # 4. Validation sur un échantillon
    pdf = df.limit(1000).toPandas()
    res = validate_schema(pdf, require_target=False)
    if not res.is_valid:
        raise ValueError(f"Invalid inference input data: {res.errors}")

    # 5. Inférence
    preds = model.transform(df)

    # 6. Rapport
    report = {
        "run_type": "inference",
        "timestamp": datetime.utcnow().isoformat(),
        "data": {
            "input_path": "s3a://nyc-interim/yellow/2023/02",
            "rows_scored": preds.count(),
        },
        ...
    }
```

7.2 Différences Training vs Inference

Aspect	Training	Inference
Target (`total_amount`)	Requis	Non requis
Validation	`require_target=True`	`require_target=False`
Feature engineering	`add_features()`	`add_features()` (identique)
Output	Modèle + métriques	Prédictions

---

8. Validation des données

8.1 Module validation.py

Colonnes requises

```
REQUIRED_COLUMNS = [
    "VendorID",
    "tpep_pickup_datetime",
    "tpep_dropoff_datetime",
    "passenger_count",
    "trip_distance",
    "RatecodeID",
    "store_and_fwd_flag",
    "PULocationID",
    "DOLocationID",
    "payment_type",
    "total_amount",  # Requis seulement pour training
]
```

Colonnes non-négatives

```
NON_NEGATIVE_COLUMNS = [
    "passenger_count",
    "trip_distance",
    "total_amount",
]
```

Fonctions de validation

Fonction	Vérification
`validate_schema()`	Colonnes requises présentes
`validate_non_negative()`	Pas de valeurs négatives
`validate_datetime_sanity()`	pickup ≤ dropoff

---

9. Tests Machine Learning

9.1 Organisation des tests

```
tests/
├── test_ml_schema.py        # Schéma des données
├── test_ml_quality.py       # Qualité du modèle
├── test_ml_plausibility.py  # Plausibilité métier
├── test_validation.py       # Validation générale
└── test_month_range.py      # Plages de mois
```

9.2 Tests de schéma (test_ml_schema.py)

Phase : Pre-training

```
class TestTrainingSchema:
    def test_all_required_columns_present(self):
        """Training data must have all required columns including target."""
        df = _create_valid_training_df()
        result = validate_schema(df, require_target=True)
        assert result.is_valid

    def test_missing_target_fails(self):
        """Training without target column must fail."""
        df = _create_valid_inference_df()
        result = validate_schema(df, require_target=True)
        assert not result.is_valid

class TestInferenceSchema:
    def test_inference_without_target_valid(self):
        """Inference data without target column must be valid."""
        df = _create_valid_inference_df()
        result = validate_schema(df, require_target=False)
        assert result.is_valid
```

9.3 Tests de qualité (test_ml_quality.py)

Phase : Post-training

```python
# Seuils de qualité
MAX_RMSE_THRESHOLD = 10.0   # Exigence du projet
MIN_R2_THRESHOLD = 0.0      # Meilleur que la moyenne
MAX_MAE_THRESHOLD = 15.0    # Raisonnable
MIN_ACCEPTABLE_R2 = 0.5     # Bon modèle

class TestRMSEQuality:
    def test_actual_model_rmse_if_available(self):
        """Test actual model RMSE from training metrics."""
        metrics = load_training_metrics()
        rmse = metrics.get("metrics", {}).get("rmse")
        assert rmse < MAX_RMSE_THRESHOLD

class TestModelQualityIntegration:
    def test_metrics_consistency(self):
        """MAE should always be <= RMSE (mathematical property)."""
        metrics = load_training_metrics()
        rmse = metrics.get("metrics", {}).get("rmse")
        mae = metrics.get("metrics", {}).get("mae")
        assert mae <= rmse
```

9.4 Tests de plausibilité métier (test_ml_plausibility.py)

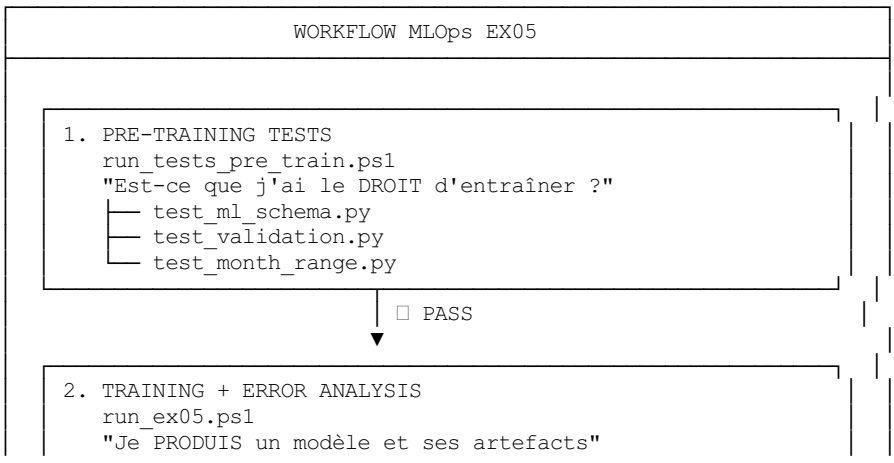
Phase : Post-prediction

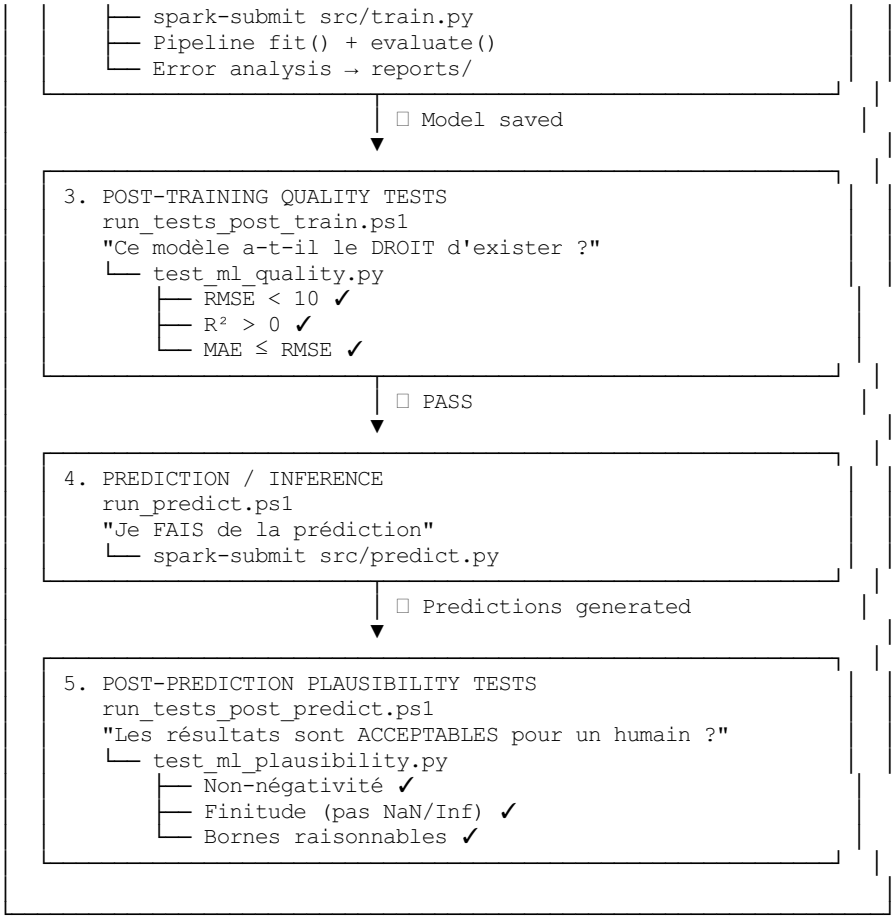
```python
# Contraintes métier NYC Taxi
MIN_FARE = 0.0              # Minimum possible
MAX_REASONABLE_FARE = 500.0  # Maximum raisonnable
TYPICAL_MIN_FARE = 2.50      # Tarif de base NYC
TYPICAL_MAX_FARE = 200.0     # Maximum typique

class TestPredictionNonNegativity:
    def test_array_predictions_non_negative(self):
        """Array of predictions should all be non-negative."""
        predictions = np.array([10.5, 25.0, 15.75, 8.0, 45.0])
        assert np.all(predictions >= MIN_FARE)

class TestPredictionFiniteness:
    def test_nan_prediction_invalid(self):
        """NaN prediction is invalid."""
        prediction = float("nan")
        assert not math.isfinite(prediction)

class TestBusinessConstraints:
    def test_airport_flat_rate_reasonable(self):
        """Airport flat rates should be within bounds."""
        jfk_flat_rate = 52.0  # JFK flat rate to Manhattan
        assert jfk_flat_rate <= MAX_REASONABLE_FARE
```

---

10. Workflow MLOps et scripts

10.1 Vue d'ensemble du workflow





10.2 Détail des scripts

Script 1 : run_tests_pre_train.ps1

```
# Tests AVANT entraînement
pytest tests/test_validation.py tests/test_month_range.py tests/test_ml_schema.py -v
```

Question : "Est-ce que j'ai le droit d'entraîner ?"

Script 2 : run_ex05.ps1

```
# Entraînement Spark
docker exec spark-master bash -c @"
cd /opt/workdir/ex05_ml_prediction_service && \
spark-submit --master spark://spark-master:7077 src/train.py
"@
```

Question : "Je produis un modèle et ses artefacts"

Script 3 : run_tests_post_train.ps1

```
# Tests qualité modèle
pytest tests/test_ml_quality.py -v
```

Question : "Ce modèle a-t-il le droit d'exister ?"

Script 4 : run_predict.ps1

```
# Inférence Spark
docker exec spark-master bash -c @"
cd /opt/workdir/ex05_ml_prediction_service && \
spark-submit --master spark://spark-master:7077 src/predict.py
"@
```

Question : "Je fais de la prédiction"

Script 5 : run_tests_post_predict.ps1

```
# Tests plausibilité métier
pytest tests/test_ml_plausibility.py -v
```

Question : "Les résultats sont acceptables pour un humain ?"

10.3 Exécution complète

```
cd ex05_ml_prediction_service

# Workflow complet
.\run_tests_pre_train.ps1      # ~30s
.\run_ex05.ps1                  # ~4000s (1h+)
.\run_tests_post_train.ps1      # ~30s
.\run_predict.ps1               # ~90s
.\run_tests_post_predict.ps1    # ~30s
```

---

11. Artefacts générés

11.1 Modèle

```
models/ex05_spark_model/
├── metadata/
│   └── part-00000           # Métadonnées du pipeline
└── stages/
    ├── 0_StringIndexer_*/   # VendorID indexer
    ├── 1_StringIndexer_*/   # RatecodeID indexer
    ├── ...
    ├── 6_OneHotEncoder_*/
    ├── ...
    ├── 12_VectorAssembler_*/
    └── 13_GBTRegressor_*/   # Modèle entraîné
```

11.2 Rapports

train_metrics.json


```
{
  "run_type": "training",
  "model": "GBTRegressor",
  "timestamp": "2026-01-04T10:30:00.000000",
  "data": {
    "months": ["2023/01", "2023/02"],
    "train_rows": 4500000,
    "test_rows": 1125000
  },
  "metrics": {
    "rmse": 5.17,
    "mae": 2.05,
    "r2": 0.94
  },
  "params": {
    "maxDepth": 6,
    "maxIter": 50,
    "seed": 42
  },
  "execution": {
    "duration_seconds": 4032.15
  },
  "error_analysis": {
    "summary": {...},
    "business_insights": [...]
  }
}
```

error_summary.json


```
{
  "rows_analyzed": 1125000,
  "summary": {
    "count": 1125000,
    "mean_error": 0.12,
    "std_error": 5.15,
    "min_error": -245.30,
    "max_error": 312.50,
    "median_error": 0.05,
    "mean_abs_error": 2.05,
    "p25_error": -1.20,
    "p75_error": 1.35,
    "p95_error": 8.50,
    "p99_error": 18.20,
    "pct_underestimate": 48.5,
    "pct_overestimate": 51.5
  },
  "business_insights": [
    "Model predictions are well-balanced between over/under estimation.",
    "High-value trips (>$60) have elevated MAE ($12.50). Consider: airport flat rates, negotiated fares.
    "Most common cause of high errors: 'airport_flat_rate' (4/10 top errors)."
  ]
}
```


error_by_price_bucket.json


```
{
  "buckets": [
    {
      "bucket_name": "low",
      "min_price": 0,
      "max_price": 10,
      "count": 280000,
      "pct_of_total": 24.89,
      "mean_error": 0.15,
      "mean_abs_error": 1.25,
      "rmse": 2.10,
      "median_error": 0.10
    },
    {
      "bucket_name": "medium",
      "min_price": 10,
      "max_price": 30,
      "count": 620000,
      "pct_of_total": 55.11,
      "mean_error": 0.08,
      "mean_abs_error": 1.85,
      "rmse": 3.20,
      "median_error": 0.05
    },
    ...
  ]
}
```


predict_report.json

```
{
  "run_type": "inference",
  "timestamp": "2026-01-04T12:00:00.000000",
  "data": {
    "input_path": "s3a://nyc-interim/yellow/2023/02",
    "rows_scored": 2850000
  },
  "validation": {
    "schema_valid": true,
    "sample_size": 1000
  },
  "execution": {
    "duration_seconds": 84.50
  }
}
```


---

12. Conformité aux exigences

12.1 Checklist du sujet

Exigence	Statut	Implémentation
Spark MLlib	☐	GBTRegressor + Pipeline
Données MinIO	☐	S3A connector
Prédire `total_amount`	☐	Target définie
RMSE < 10	☐	RMSE = 5.17
Feature engineering	☐	src/features.py
Anti-leakage	☐	Colonnes monétaires supprimées
Pas de notebooks	☐	Scripts Python uniquement
PEP8	☐	`.flake8` configuré
NumpyDoc	☐	Documentation complète
Tests unitaires	☐	5 fichiers pytest
Reproductibilité	☐	seed=42, Docker

12.2 Bonnes pratiques MLOps implémentées

Pratique	Implémentation
Versionnement du modèle	`models/ex05_spark_model/`
Traçabilité	`train_metrics.json` avec timestamp
Reproductibilité	seed fixe, Docker
Tests automatisés	pytest + scripts PowerShell
Analyse d'erreurs	Module dédié avec insights métier
Validation des données	Pre-training + pre-inférence
Séparation des responsabilités	Modules distincts (train, predict, features)
Configuration centralisée	`config.py`

12.3 Performances

Métrique	Valeur
Temps d'entraînement	~4032s (1h07)
Temps d'inférence	~84s
Données entraînement	~5.6M lignes
Données test	~1.1M lignes
Taille modèle	~500 MB

Conclusion

L’exercice 05 implémente un **pipeline Machine Learning complet et professionnel** dans un contexte Big Data, respectant les standards MLOps :

1. **Architecture modulaire** : Chaque responsabilité est isolée dans un module dédié
2. **Workflow structuré** : 5 étapes avec validation à chaque phase
3. **Qualité du code** : PEP8, NumpyDoc, tests automatisés
4. **Analyse approfondie** : Error analysis avec justifications métier
5. **Reproductibilité** : Docker, seeds fixes, artefacts versionnés
6. **Performance** : RMSE = 5.17 (bien sous le seuil de 10)

Le projet est prêt pour une **mise en production** ou une **présentation académique**.

Auteur : MAAOUIA Ahmed – CY Tech Big Data
Date : Janvier 2026
Version : 1.0