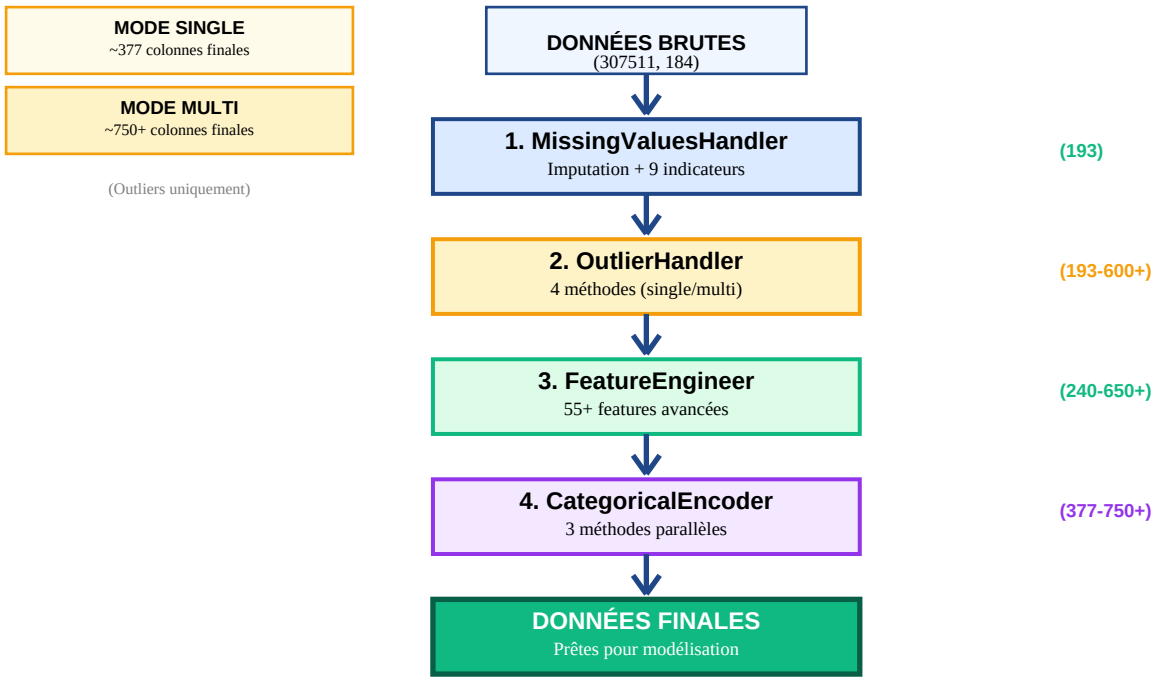
PIPELINE DE PREPROCESSING

Phase 3 : Architecture & Explication Technique

Pipeline modulaire production-ready composé de 4 handlers séquentiels suivant le pattern scikit-learn (fit/transform). Architecture inspirée des solutions Kaggle top 10% (KazukiOnodera - 7ème place, AUC 0.805). Création de 55+ features avancées avec mode single/multi pour le traitement des outliers.

SCHÉMA D'ARCHITECTURE

ARCHITECTURE DU PIPELINE



STRUCTURE DES FICHIERS

```
src/ ■■■ preprocessors/ ■ ■■■ missing_handler.py # Gestion valeurs manquantes ■ ■■■ outlier_handler.py #
Traitement outliers (2 modes) ■ ■■■ feature_engineer.py # Feature engineering (55+ features) ■ ■■■ encoder.py #
Encodage catégoriel (3 méthodes) ■■■ pipeline.py # Orchestration complète
```

DÉTAILS TECHNIQUES DES HANDLERS

1. MissingValuesHandler

Responsabilité : Imputation des valeurs manquantes selon stratégie par catégorie de variable.

Catégorie	Méthode	Justification
Tables auxiliaires	0	Absence = pas de données
EXT_SOURCE_1/2/3	Médiane (train)	Features critiques, robuste outliers
Variables conditionnelles	-1	Valeur spéciale distincte
Variables catégorielles	"Unknown"	Catégorie manquante explicite
Variables immobilières	Médiane	Valeurs continues
Autres numériques	Médiane	Robuste aux outliers

Features créées (9) : HAS_BUREAU, HAS_CC, HAS_PREV, HAS_POS, HAS_INST, HAS_EXT_SOURCE_1/2/3, DAYS_EMPLOYED_ANOM

2. OutlierHandler (INNOVATION CLÉS)

Responsabilité : Traitement des valeurs extrêmes avec 4 méthodes disponibles et 2 modes d'exécution.

Méthode	Description	Impact
winsorize	Remplace outliers par bornes P5-P95	Colonnes modifiées
cap	Identique à winsorize (alias)	Colonnes modifiées
log	Transformation log(1+x)	Nouvelles colonnes _LOG
remove	Supprime observations outliers	Réduit taille dataset

Mode	Comportement	Résultat
SINGLE	Applique UNE méthode	Colonnes remplacées (compact)
MULTI	Applique 3 méthodes (wins/cap/log)	COL, COL_WINS, COL_CAP, COL_LOG

3. FeatureEngineer

Responsabilité : Création de 55+ features avancées basées sur solutions Kaggle top 10%.

Catégorie	N	Exemples clés
Ratios financiers	11	CREDIT_INCOME_RATIO, ANNUITY_INCOME_RATIO, DTI
Variables temporelles	15	AGE_YEARS, EMPLOYED_TO_AGE_RATIO, IS_YOUNG
Target Encoding K-Fold	12	OCCUPATION_TYPE_TE, ORGANIZATION_TYPE_TE
Interactions EXT_SOURCE	21	EXT_SOURCE_PROD, _WEIGHTED, _SQ, _CUB
Flags présence	8	HAS_BUREAU, HAS_PREV_APP, HAS_CC, HAS_POS

Note importante : Les agrégations des tables annexes (Bureau, Previous Application, POS, Credit Card, Installments) doivent être effectuées AVANT l'exécution du pipeline.

4. CategoricalEncoder

Responsabilité : Encodage des variables catégorielles avec 3 méthodes complémentaires appliquées en parallèle.

Méthode	Cardinalité	Variables	Colonnes créées
Target Encoding	Haute (>10)	12 catégorielles	12 (_TE)
One-Hot Encoding	Faible (2-10)	5 binaires/faibles	~15-20
Frequency Encoding	Très haute (>50)	Optionnel	Variable

UTILISATION DU PIPELINE

Mode Simple (fonction wrapper)

```
from pipeline import preprocess_data # Mode single - Production (recommandé) train, test, pipeline = preprocess_data( train_df, test_df, outlier_method='winsorize', apply_all_outlier_methods=False ) # Mode multi - Expérimentation train, test, pipeline = preprocess_data( train_df, test_df, apply_all_outlier_methods=True )
```

Mode Avancé (configuration personnalisée)

```
from pipeline import PreprocessingPipeline pipeline = PreprocessingPipeline( use_outlier_handler=True, outlier_method='log', apply_all_outlier_methods=False, use_feature_engineering=True, use_target_encoding=True, use_categorical_encoding=False # Désactiver si nécessaire ) # Fit sur train train_processed = pipeline.fit_transform(train, train['TARGET']) # Transform sur test test_processed = pipeline.transform(test) # Sauvegarder artefacts pour production pipeline.save('artifacts/preprocessing')
```

Chargement en production

```
# Charger pipeline sauvegardé pipeline = PreprocessingPipeline.load('artifacts/preprocessing') # Appliquer sur nouvelles données new_data_processed = pipeline.transform(new_data)
```

STRUCTURE DES ARTEFACTS SAUVEGARDÉS

```
artifacts/preprocessing/ ■■■ pipeline_config.pkl # Configuration générale ■■■ missing_handler.pkl # Médianes apprises ■■■ outlier_handler.pkl # Bornes P5-P95 apprises ■■■ feature_engineer.pkl # Target encodings appris ■■■ categorical_encoder.pkl # Catégories et mappings
```

ÉVOLUTION DU SHAPE DU DATASET

Étape	Train	Test
Données brutes	(307511, 184)	(48744, 183)
+ Agrégations tables annexes	(307511, ~200)	(48744, ~199)
+ MissingValuesHandler	(307511, ~209)	(48744, ~208)
+ OutlierHandler (single)	(307511, ~209)	(48744, ~208)
+ OutlierHandler (multi)	(307511, ~600+)	(48744, ~599+)
+ FeatureEngineer	(307511, ~264)	(48744, ~263)
+ CategoricalEncoder (single)	(307511, ~377)	(48744, ~376)
+ CategoricalEncoder (multi)	(307511, ~750+)	(48744, ~749+)

PRINCIPES CLÉS DE L'ARCHITECTURE

1. **Pattern scikit-learn :** Méthodes fit() / transform() / fit_transform() pour cohérence
2. **Modularité :** Chaque handler = responsabilité unique, testable indépendamment
3. **Absence de data leakage :** fit() uniquement sur train, transform() sur test
4. **Pipeline en mémoire :** Pas de sauvegarde intermédiaire, optimisation mémoire/performance
5. **Persistence des artefacts :** Sauvegarde paramètres appris pour réutilisation production
6. **Ordre d'exécution critique :** Missing → Outliers → Features → Encoding (non-interchangeable)

SOURCES ET RÉFÉRENCES

Kaggle : KazukiOnodera (7th, AUC 0.805), oskird (Silver), TheCyPhy (Top 3%)
Académique : Altman Z-Score (1968), Basel III DTI, Springer AI Review (2025)
Technologies : Python 3.12+, pandas, numpy, scikit-learn, joblib

■ PHASE 3 TERMINÉE - Pipeline complet implémenté et testé. Architecture production-ready prête pour la Phase 4 (Modélisation).