MSPR3 : Big Data & Analyse de Données - Dossier de Synthèse

# Présentation de l’entreprise et de son activité

Notre client est une start-up de conseil en campagnes électorales fondée par **Jean-Edouard de la Motte Rouge**. L’équipe cœur réunit un·e expert·e en analyse politique, un·e business developer et un·e assistant·e. L’ambition est de prédire les tendances électorales à venir grâce à l’IA, en s’appuyant sur des indicateurs factuels tels que **sécurité, emploi, vie associative, population, tissu économique (entreprises), pauvreté**, etc. Cette capacité de projection constitue un **avantage concurrentiel** décisif pour mieux cibler les messages, optimiser l’allocation des moyens de campagne et conseiller les candidats/partis sur leurs territoires prioritaires.

Avant d’investir plus largement (infrastructure, R&D, demandes d’aides à l’innovation), l’entreprise mandate notre équipe pour **réaliser une preuve de concept (POC)** sur un périmètre restreint et unique.

# Contexte & objectifs

L'objectif de ce projet est de réaliser une preuve de concept (POC) pour la start-up de M. de la Motte Rouge. Cette POC vise à prédire le parti politique qui arrivera en tête lors des élections sur un territoire géographique donné, en se basant sur des données électorales passées et des indicateurs socio-économiques.

Ce dossier retrace la démarche suivie, les choix effectués et les résultats obtenus.

# Choix du périmètre et des données

## Zone géographique

Le périmètre du POC est limité à **Nantes Métropole (EPCI 244400404)**. Ce choix n’est pas arbitraire : il répond à un double impératif de **pertinence analytique** et de **faisabilité opérationnelle**.

D’un point de vue politique, la métropole constitue un territoire représentatif des recompositions électorales contemporaines. On y retrouve à la fois une ville-centre fortement marquée à gauche, des communes périphériques davantage ouvertes aux écologistes, mais aussi des zones périurbaines où la droite traditionnelle ou l’extrême-droite progressent. Cette diversité interne permet de tester un modèle prédictif sur un spectre électoral complet, sans sortir d’un même bassin de vie.

Sur le plan socio-économique, l’intercommunalité regroupe des réalités contrastées : quartiers urbains denses, zones pavillonnaires, territoires en forte croissance démographique, communes populaires confrontées à la précarité, pôles d’emploi dynamiques et espaces résidentiels plus stables. Ces différences se traduisent dans les indicateurs que nous collectons (revenus, chômage, pauvreté, démographie, tissu économique et associatif), offrant une variabilité suffisante pour établir des corrélations robustes entre contexte local et comportement électoral.

Le choix de l’échelle communale s’explique par sa pertinence technique et stratégique. C’est la maille la plus fine où les résultats électoraux sont systématiquement publiés et exploitables, tout en restant anonymes et conformes au RGPD. Elle se prête à des décisions concrètes pour une campagne (allocation de moyens, ciblage des messages, priorisation des zones), et se joint parfaitement avec les bases statistiques nationales (INSEE, FILOSOFI, RP, SIRENE, SSMSI). Nantes Métropole, avec ses 24 communes stables et bien documentées, constitue donc un compromis idéal : assez vaste pour être représentatif, assez limité pour être gérable.

Enfin, ce choix est également stratégique : valider la méthode sur Nantes Métropole prépare une montée en échelle future. Le pipeline construit pour ce territoire pourra être répliqué à d’autres EPCI, départements ou régions, sans réécrire l’architecture. Nantes Métropole est donc un **banc d’essai exigeant mais maîtrisable**, garantissant que le POC reste crédible tout en étant réalisable dans le cadre temporel et académique fixé.

## Choix et justification des données

Pour répondre à l’objectif de prédiction, deux familles de données sont mobilisées : les **résultats électoraux** et les **indicateurs socio-politiques de contexte**.

Les résultats électoraux constituent la cible et la variable dépendante principale. Nous retenons les scrutins présidentiels (2012, 2017, 2022), législatifs (2012, 2017, 2022), européens (2014, 2019) et municipaux (2014, 2020), en intégrant les deux tours lorsque la mécanique électorale l’exige. Ce choix permet de couvrir des élections de nature différente : les présidentielles structurent les grandes familles partisanes, les législatives révèlent des dynamiques locales, les européennes mesurent des comportements d’adhésion ou de contestation, et les municipales éclairent l’impact de la proximité et des notables. Pour chaque scrutin, nous collectons systématiquement inscrits, votants, abstention, blancs, nuls et exprimés, puis les voix et pourcentages par candidat ou liste. Ces données, harmonisées par des règles de nommage strictes (slugs sans accents, minuscules, séparateurs standardisés), permettent d’identifier de façon claire le **parti en tête**, qui constitue notre variable cible.

En complément, des indicateurs socio-politiques sont intégrés comme variables explicatives. Nous retenons la population municipale pour contextualiser les volumes, le revenu médian et le taux de pauvreté issus de FILOSOFI pour qualifier le niveau de vie, le taux de chômage et la structure d’âge pour refléter le rapport au marché du travail et les dynamiques générationnelles, des indicateurs sur le logement (vacance, logements sociaux) et la densité pour capturer la morphologie urbaine, des données économiques (entreprises actives, créations) et sociales (associations) pour appréhender le dynamisme local, ainsi qu’un proxy de délinquance pour refléter les préoccupations sécuritaires. Chaque indicateur est sourcé (INSEE, SIRENE, RNA, SSMSI), millésimé et documenté. Lorsqu’un décalage existe entre l’année de publication et l’année du scrutin, il est explicité dans le dictionnaire et le README afin de maintenir la transparence.

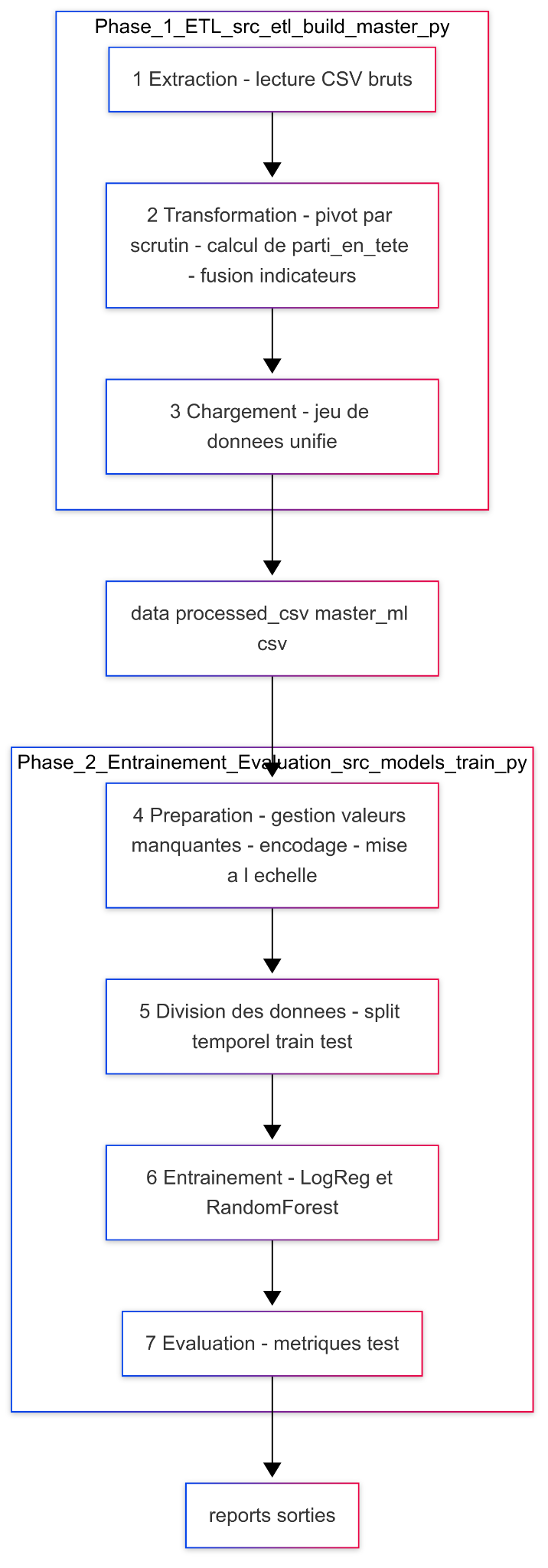
La normalisation de ces données suit des règles strictes : format CSV UTF-8 homogène, clés de jointure systématiques, typage clair des colonnes, complétude minimale de 95 %, contrôles automatiques des ratios (par exemple, que la somme des pourcentages de voix avoisine 100 %). Les éventuelles données estimées sont explicitement signalées et exclues de l’entraînement par défaut.

| Année | Critères | Source |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Démarche et Méthodologie

## Schéma de Traitement des Données (Flux)

Le flux de données suit une approche ETL (Extract, Transform, Load) classique pour préparer les données brutes à l’analyse et à la modélisation



## Architecture et outils

### Vision d’ensemble

Notre objectif d’architecture est double : **reproductibilité** (quiconque peut relancer le POC à l’identique) et **portabilité** (exécutable sur n’importe quelle machine sans problème d'environnement). Pour cela, on s’appuie sur **Python 3.11**, un socle **Docker / Docker Compose**, un dépôt **GitHub** pour le code et la gestion de versions, et **Google Drive** pour héberger les jeux de données et livrables lourds (afin de garder un dépôt git léger). L’arborescence standardisée et les commandes d’exécution sont décrites dans le README du projet (données brutes dans data/raw\_csv/, jeu ML final dans data/processed\_csv/, rapports dans reports/, code dans src/)

### Code & gouvernance (Github)

Nous avons centralisé **le code source** dans un dépôt GitHub privé. L’organisation est volontairement simple : branche main pour les versions stables, branches de **feature** pour les évolutions ponctuelles, et **pull requests** pour la revue. Les **issues** servent à suivre les tâches (ETL, ajout d’indicateurs, entraînement, cartes). Les données volumineuses ne sont **pas** versionnées dans git : seules les **spécifications** (README, scripts, dictionnaire des données) vivent dans le dépôt, ce qui évite les historiques trop lourds et les conflits sur fichiers binaires. Les chemins et scripts référencés (ETL, entraînement, export cartes) correspondent au plan exposé dans le dossier et dans le README.

### Environnement d’exécution (Docker / Compose)

Tout le projet est **conteneurisé** pour garantir la même pile logicielle chez tout le monde : image Python 3.11 avec dépendances **pandas / scikit-learn / matplotlib / requests** épinglées. On utilise **Docker Compose** pour monter les volumes ./data, ./reports, ./docs dans le conteneur, puis lancer les commandes “métier” :

* **Build** de l’image puis **ETL** → master\_ml.csv :  
   docker compose build puis  
   docker compose run --rm app src/etl/build\_master.py --raw-dir data/raw\_csv --out data/processed\_csv/master\_ml.csv
* **Entraînement** (par défaut la dernière année est utilisée comme test, ou bien années de test explicites) :  
   docker compose run --rm app src/models/train.py --data data/processed\_csv/master\_ml.csv  
   docker compose run --rm app src/models/train.py --data data/processed\_csv/master\_ml.csv --test-years 2020 2022

Ces commandes, ainsi que les **raccourcis Makefile** et le **pipeline Compose** pour enchaîner les étapes, sont décrits et maintenus dans le README.  
 Le flux logique (ETL → entraînement → rapports/figures) est le même que celui schématisé dans le dossier de synthèse.

### Architecture des données

L’architecture est orientée “lacs de CSV normalisés” + un **jeu maître** unique pour le ML :

* **data/raw\_csv/** : dépôts des fichiers **bruts normalisés** (élections par commune et par tour, indicateurs socio-éco, référentiel communes, dictionnaire des nuances).
* **data/processed\_csv/** : résultats de l’ETL, notamment **master\_ml.csv** qui sert directement à l’entraînement.
* **reports/** : sorties d’évaluation (métriques, matrices de confusion, cartes).
* **src/etl/** et **src/models/** : scripts de transformation et d’entraînement.  
   Cette structuration et les fichiers attendus sont rappelés dans le README, avec les effets de chaque script.

### Outils Python (3.11)

Côté Python, on reste **sobre et standard** :

* **pandas** pour l’ingestion/normalisation,
* **scikit-learn** pour la préparation (encodage, standardisation) et la modélisation (**LogisticRegression** et **RandomForestClassifier**),
* **matplotlib** pour les figures (matrices de confusion) et l’export des cartes (via un script qui lit un GeoJSON des communes).  
   Les choix de modèles et la logique d’évaluation sont décrits dans le dossier de synthèse et reflétés dans src/models/train.py

### Stockage externe (Google Drive)

Pour séparer **code** et **données lourdes / livrables**, nous avons organisé un espace **Google Drive** avec une arborescence simple. L’idée est que l’étudiant·e qui reprend le projet récupère tout **sans** re-scraper, et que les évaluateurs aient un accès direct aux livrables.

**Contenu du Drive (proposé et utilisé dans le projet) :**

* **01\_sources\_brutes/** Archives originales **telles que téléchargées** (data.gouv.fr / INSEE / Ministère de l’Intérieur) : résultats par commune (\*\_par\_commune.csv), référentiel des communes de Nantes Métropole, **indicateurs socio-éco** (FILOSOFI, RP, SIRENE/RNA), et, si présent, le GeoJSON des communes (ou la note expliquant le fetch automatique).
* **02\_csv\_normalises/** Les mêmes données mises au format du projet (UTF-8, virgule, en-têtes), prêtes à être copiées dans data/raw\_csv/. On y place aussi **data\_dictionary.csv** et le **README.md** d’extraction/normalisation.
* **03\_processed/** Le **master\_ml.csv** produit par l’ETL et, si utile, un export parquet ou zip complémentaire.
* **04\_docs/** Le **dossier de synthèse** (mspr.md / PDF), le **template de slides** (Markdown et/ou PDF), le **quality\_report.md** (complétude, checks), et le **provenance.csv** (URL source, date d’extraction, licence, SHA256).
* **05\_reports/** Les **métriques** (metrics.csv), les **matrices de confusion** et les **cartes PNG** exportées par les scripts.
* **06\_releases/** Les **archives ZIP** prêtes à diffusion (ex. le projet Docker packagé), afin de partager une version “prête à lancer”.

Cette séparation a trois avantages : elle évite d’alourdir Git, elle **pérennise** les sources utilisées (auditabilité), et elle réduit le temps de prise en main pour un tiers (copier/coller les répertoires 02\_csv\_normalises → data/raw\_csv/, lancer Compose). Les noms de dossiers ci-dessus reflètent les artefacts mentionnés dans le dossier et le README (packs CSV, master, rapports).

## Modèle Conceptuel de Données (MCD)

### Qu’est-ce qu’un MCD ?

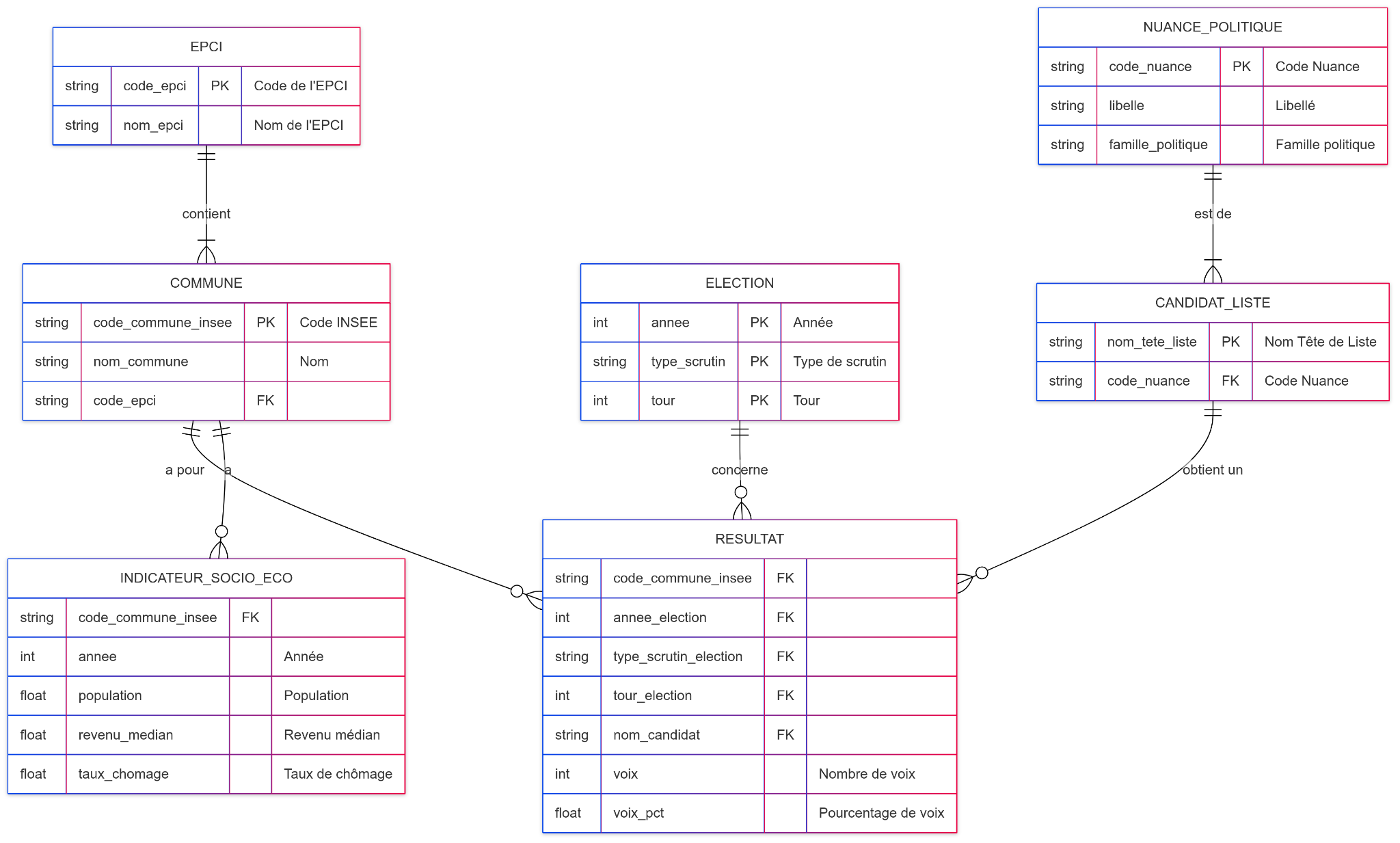
Le **MCD** décrit, au niveau conceptuel, **les objets métier** (entités), **leurs attributs clés** et **leurs relations** (cardinalités), indépendamment de tout format technique. C’est la carte d’identité de la donnée : elle fige un vocabulaire commun, évite les ambiguïtés et sert de base aux contrôles de qualité comme aux jointures.

Notre modèle pivote autour de la **commune** (maille d’analyse) et d’un **scrutin** (année, type, tour).

* **EPCI** contient des **COMMUNE** : Nantes Métropole est notre périmètre ; la commune (code INSEE) est la clé pivot de toutes les jointures.
* **ELECTION** identifie chaque scrutin par **(année, type\_scrutin, tour)** ; cela évite de mélanger les tours et simplifie l’évaluation temporelle.
* **NUANCE\_POLITIQUE** normalise les familles partisanes (code, libellé, famille).
* **CANDIDAT\_LISTE** représente la tête de liste ou le candidat associé à une **nuance**.
* **RESULTAT** est l’observation élémentaire **commune × élection × candidat** : on y stocke les **voix** et **voix\_pct**, ce qui permet de reconstituer le **parti/nuance en tête** et de vérifier la cohérence des parts.
* **INDICATEUR\_SOCIO\_ECO** décrit chaque **commune × année** (population, revenu médian, chômage, etc.) pour relier contexte et vote.

Les cardinalités suivent la logique métier : un EPCI **contient** plusieurs communes ; une élection **concerne** de nombreux résultats communaux ; un candidat **obtient** des voix dans ces résultats ; une nuance **classifie** les candidats ; une commune **a** ses indicateurs par année.

### Schéma



## Nettoyage et Préparation des Données (ETL)

L’ETL a un but unique : transformer nos CSV bruts hétérogènes en un **fichier maître unique et propre**, prêt pour l’apprentissage automatique : data/processed\_csv/master\_ml.csv. Toute la logique est scriptée dans src/etl/build\_master.py et s’exécute dans l’environnement Docker pour garantir la reproductibilité.

Les fichiers bruts sont déposés dans data/raw\_csv/ (référentiel des communes de Nantes Métropole, nuances politiques, résultats d’élections par commune, indicateurs socio-économiques annuels). Le script lit ces sources en UTF-8 avec repli si besoin, uniformise les entêtes en *snake\_case*, force les types (années et tours en entiers, dates en ISO), et pad les codes INSEE en cinq caractères. Le périmètre est verrouillé sur l’EPCI **244400404** afin de ne garder que les communes de Nantes Métropole.

La préparation recalcule les grandeurs de participation pour sécuriser les comptes (abstention = inscrits − votants, exprimes = votants − blancs − nuls) et élimine ou marque toute incohérence manifeste (négatifs, dépassements). Les noms de candidats ou listes sont harmonisés via des *slugs* sans accents et en minuscules, ce qui permet de créer des colonnes de voix et de pourcentages stables dans le temps. Les ratios standards sont produits de façon systématique : turnout\_pct = votants/inscrits, blancs\_pct = blancs/votants, nuls\_pct = nuls/votants, et voix\_pct\_<slug> = voix\_<slug>/exprimes. À partir de ces éléments, la **cible** de notre POC est dérivée au niveau commune × scrutin : parti\_en\_tete, candidat\_en\_tete, voix\_en\_tete, voix\_pct\_en\_tete.

Le contexte est injecté par une jointure stricte sur (code\_commune\_insee, annee) avec le fichier des indicateurs (indicateurs\_2012\_2022.csv). Aucune imputation automatique n’est réalisée : une valeur manquante reste vide ; si une estimation est nécessaire, elle est explicitement marquée (estime=true avec la méthode), et ces lignes sont exclues par défaut lors de l’entraînement. Lorsque des séries le permettent, des deltas temporels par commune et type de scrutin sont calculés (par exemple une variation de revenu médian) afin d’enrichir les futures features.

Des contrôles de qualité sont exécutés à la fin du pipeline : unicité de la clé (code\_commune\_insee, type\_scrutin, annee, tour), bornes logiques des taux dans l’intervalle [0;1], somme des pourcentages de voix au voisinage de 100 % et vérification de la complétude des colonnes clés. L’export applique un format strict et homogène (CSV UTF-8, séparateur virgule, entête première ligne) et garantit la présence des clés de jointure dans toutes les lignes (code\_commune\_insee, nom\_commune, code\_epci, annee, type\_scrutin, tour, date\_scrutin).

L’exécution se fait de manière déterministe dans le conteneur. Une version “pas à pas” consiste à construire l’image, puis lancer l’ETL :

docker compose build

docker compose run --rm app src/etl/build\_master.py --raw-dir data/raw\_csv --out data/processed\_csv/master\_ml.csv

La version “automatique” enchaîne ETL, entraînement et cartes dès le démarrage :

docker compose up -d --build

docker compose logs -f auto

En résumé, cet ETL aligne toutes les sources sur les mêmes clés, sécurise les comptes et les ratios, calcule la cible de classification, rattache le contexte socio-économique et livre un seul jeu de données maître pour la suite du POC.