Rapport d'Analyse : Infrastructure IT et Machine Learning

Projet Data Science - Villes Indiennes

3 décembre 2025

Table des matières

1 Introduction

Ce jeu de données traite de la transformation numérique et de la modernisation des infrastructures urbaines en Inde, en analysant l'évolution de 30 villes majeures sur la période 2019-2024. La problématique centrale consiste à évaluer la progression des initiatives *« Smart City »* à travers des indicateurs clés tels que l'accès à Internet, la couverture haut débit (3G/4G), et l'adoption de technologies de gestion intelligente (compteurs d'eau et d'électricité, transports dynamiques), afin de mesurer la maturité digitale de ces métropoles et d'identifier les disparités technologiques régionales.

2 Explication du Code et des Étapes de Machine Learning

Ce document décompose le script Python fourni pour expliquer comment chaque ligne de code s'inscrit dans un cycle de vie standard de projet de Data Science, de l'ingestion des données à l'évaluation des modèles.

2.1 Importation des Librairies et Chargement des Données

Avant tout traitement, il est nécessaire de charger les outils. Le code importe des bibliothèques spécifiques, chacune ayant un rôle précis dans la chaîne de valeur du Machine Learning.

Le Code :

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# ... (imports sklearn voir sections suivantes)
df = pd.read_csv('/kaggle/input/.../ICT_Subdimension_Dataset.csv')
```

L'Explication :

- `pandas` : C'est la structure vertébrale. Elle charge le fichier CSV dans un objet appelé `DataFrame` (`df`), qui est un tableau manipulable (lignes/colonnes) en mémoire.
- `seaborn` / `matplotlib` : Ce sont les outils de visualisation. Le Machine Learning commence toujours par « voir » les données pour comprendre ce que l'on va modéliser.

2.2 Analyse Exploratoire des Données (EDA)

C'est l'étape de diagnostic. On ne peut pas entraîner un modèle sur des données que l'on ne comprend pas.

Le Code :

```
df.head()
df.info()
df.describe()
sns.barplot(...)
```

L'Explication :

— `df.info()` : Permet de repérer les types de données (ex : est-ce que « Année » est un nombre ou du texte ?) et surtout de détecter les **valeurs manquantes (Null values)**. Les algorithmes de ML détestent les vides.
— `df.describe()` : Vérifie la distribution statistique (moyenne, min, max). Si une colonne a une valeur max aberrante (ex : 500% d'accès internet), cela faussera le modèle.
— **Visualisation (`sns.barplot`)** : Permet de vérifier les hypothèses (ex : « L'accès internet augmente-t-il avec le temps ? »). Si les graphiques ne montrent aucune tendance, le modèle aura du mal à apprendre quoi que ce soit.

2.3 Prétraitement des Données (Preprocessing)

C'est l'étape la plus critique. Les algorithmes mathématiques ne comprennent que les chiffres et sont sensibles aux échelles. Vos imports `sklearn` préparent cette étape.

2.3.1 A. Gestion des Données Manquantes

Concept : Si une cellule est vide, le calcul plante. Il faut la remplir (imputation).
Le Code :

```
from sklearn.impute import SimpleImputer
```

Explication : `SimpleImputer` remplace les trous par une stratégie définie (ex : la moyenne de la colonne).

2.3.2 B. Encodage des Variables Catégorielles

Concept : La machine ne comprend pas le mot « Mumbai ». Elle comprend des vecteurs.
Le Code :

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
```

Explication :
— `LabelEncoder` : Transforme « Mumbai », « Delhi » en 0, 1. (Attention : cela peut induire une hiérarchie fausse $0 < 1$).
— `OneHotEncoder` : Crée des colonnes binaires (Est_Mumbai : 1, Est_Delhi : 0). C'est souvent préférable pour ne pas biaiser le modèle.

2.3.3 C. Normalisation (Scaling)

Concept : Une variable en pourcentage (0-100) est « petite » comparée à une population (0-10,000,000). Le modèle va négliger le pourcentage à tort.
Le Code :

```
from sklearn.preprocessing import StandardScaler
```

Explication : `StandardScaler` ramène toutes les colonnes à la même échelle (moyenne $= 0$, écart-type $= 1$). Ainsi, le taux d'internet a autant de « poids » que le nombre de compteurs d'eau.

2.4 Construction de la Pipeline

Concept : Pour éviter d'appliquer ces transformations manuellement à chaque fois (et risquer des erreurs), on crée un « tuyau » (pipeline) automatisé.

Le Code :

```
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
```

L'Explication :

— `Pipeline` : Enchaîne les actions séquentiellement. Ex : Imputer → Scaler → Model.

— `ColumnTransformer` : Permet d'appliquer des traitements différents selon les colonnes (ex : `OneHotEncoder` sur la colonne 'City', mais `StandardScaler` sur la colonne 'Internet Access').

2.5 Division des Données (Splitting)

Concept : Il ne faut jamais tester un élève sur les exercices qu'il a déjà vus en cours. De même, on cache une partie des données au modèle pour vérifier s'il « apprend » vraiment ou s'il « par cœurise » (overfitting).

Le Code :

```
from sklearn.model_selection import train_test_split
```

L'Explication : Cette fonction divise le `df` en deux :

1. **Train Set (80%)** : Pour l'entraînement.
2. **Test Set (20%)** : Pour l'évaluation finale.

2.6 Choix et Entraînement des Modèles

Concept : Choisir l'algorithme mathématique qui va trouver la relation entre X (les infrastructures) et y (la cible, ex : le niveau de développement).

Le Code :

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

L'Explication :

— `LogisticRegression` : C'est un modèle de classification linéaire. Il est simple, rapide et interprétable. Il trace une ligne droite pour séparer les classes (ex : Ville Smart vs Ville non-Smart).

— `SVC` (Support Vector Classifier) : C'est un modèle plus puissant capable de tracer des frontières complexes (courbes) entre les catégories. Il est utile si la distinction entre les villes n'est pas évidente linéairement.

2.7 Évaluation de la Performance

Concept : Comment savoir si le modèle est bon ? On compare ses prédictions avec la réalité du Test Set.

Le Code :

```
from sklearn.metrics import matthews_corrcoef
```

L'Explication :

— Vous avez choisi le **coefficient de corrélation de Matthews (MCC)**.
— C'est une métrique très robuste pour la classification binaire.
— Si vous avez 95 villes « Smart » et 5 « Non-Smart », un modèle qui dit « Tout le monde est Smart » aurait 95% de précision (ce qui est trompeur).
— Le **MCC** prend en compte les Vrais Positifs, Vrais Négatifs, Faux Positifs et Faux Négatifs. Un score de +1 est parfait, 0 est aléatoire. C'est un choix excellent pour des datasets déséquilibrés.

3 Conclusion

L'analyse détaillée de ce script met en évidence une méthodologie rigoureuse qui dépasse la simple observation des données. Le code ne sert pas uniquement à visualiser l'état des infrastructures en Inde (via l'EDA), mais il prépare un environnement complet pour l'analyse prédictive.

Les points forts de cette approche technique sont :

— **L'Industrialisation du Processus :** L'utilisation de `Pipeline` et `ColumnTransformer` garantit que le traitement des données est standardisé, reproductible et sans fuite d'information (*Data Leakage*).
— **La Robustesse de l'Évaluation :** En choisissant le coefficient de Matthews (MCC) plutôt qu'une simple précision, le code privilégie la fiabilité du diagnostic, même face à des classes déséquilibrées.
— **La Complémentarité des Modèles :** L'importation conjointe de la `LogisticRegression` (interprétabilité) et du `SVC` (complexité) permet de tester différentes hypothèses sur la nature des données.

En somme, ce script transforme une base de données descriptive en un véritable outil d'aide à la décision. Une fois exécuté avec une variable cible définie (par exemple : prédire si une ville atteindra un statut « 100% Connecté » l'année suivante), il permettra d'identifier les facteurs clés de succès de la transformation numérique urbaine.