Sales prediction

Présentation du Projet d’Apprentissage Automatique

Houssem Brari & Jalel Eddine Hleli G3

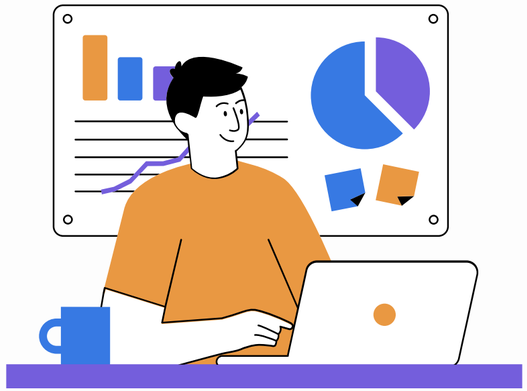
1. INTRODUCTION

PRÉSENTATION GÉNÉRALE DU PROJET

Ce projet a pour objectif de prédire les ventes à l’aide de techniques d’apprentissage automatique. En analysant un ensemble de données contenant des informations sur les produits et leurs ventes, nous cherchons à créer un modèle précis et efficace.

Objectifs du tutoriel

Préparer les données pour la modélisation.



Implémenter et entraîner un modèle de régression.

Interpréter les résultats du modèle et proposer des améliorations.

2. CADRE GÉNÉRAL DU PROJET

Ce projet appartient au domaine du commerce et de la distribution. Il permet d’anticiper les ventes, ce qui est essentiel pour une planification optimale des stocks, des campagnes marketing, et des ressources logistiques.

Importance et Pertinence du Projet

Réduisent les coûts liés à la surproduction ou aux ruptures de stock.

Aident à identifier des tendances de consommation.

Contribuent à une meilleure allocation des ressources.

3. PROBLÉMATIQUE À RÉSOUDRE

Comment construire un modèle d’apprentissage automatique capable de prédire avec précision les ventes en fonction de divers paramètres, comme les caractéristiques des produits et les prix ?

Défis Techniques et Méthodologiques

- **Disponibilité et qualité des données.**
- **Relations complexes entre variables.**
- **Surapprentissage, qui pourrait limiter la généralisation.**

4. CHOIX DU DATASET

Le dataset doit :

- **Contenir des variables pertinentes comme les caractéristiques des produits et les données de ventes.**
- **Être de taille suffisante pour assurer un entraînement efficace.**
- **Être de qualité, avec des données complètes et cohérentes.**

Présentation du Dataset Choisi

Le dataset inclut des colonnes clés comme :

- *Outlet_Type : Type de point de vente*
- *Item_MRP : Prix maximum de vente de l'article*
- *Item_Visibility : Visibilité de l'article*
- *Item_Type : Type de produit*
- *Outlet_Establishment_Year : Année de création du point de vente ...*

```
df.head()
```

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales
FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700

Source du Dataset

Kaggle

5. ANALYSE DES DONNÉES (EDA)

- **Dimensions et types de données :**

Voici un aperçu du dataset avec ses dimensions et types de colonnes

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 8523 entries, FDA15 to DRG01
Data columns (total 11 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Item_Weight                7060 non-null   float64
 1   Item_Fat_Content           8523 non-null   object
 2   Item_Visibility            8523 non-null   float64
 3   Item_Type                  8523 non-null   object
 4   Item_MRP                   8523 non-null   float64
 5   Outlet_Identifier          8523 non-null   object
 6   Outlet_Establishment_Year  8523 non-null   int64
 7   Outlet_Size                6113 non-null   object
 8   Outlet_Location_Type       8523 non-null   object
 9   Outlet_Type                8523 non-null   object
 10  Item_Outlet_Sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(6)
```

- **Dimensions : 8523 lignes , 11 colonnes**
- **Types de variables : Variables numériques et catégoriques.**

Prétraitement des Données

Gestion des valeurs manquantes : La distribution des valeurs manquantes dans les colonnes du dataset est la suivante :

Analyse:

La colonne Item_Weight présente 1463 valeurs manquantes, et la colonne Outlet_Size en a 2410. Ces valeurs doivent être prises en compte pour décider des méthodes de traitement appropriées (imputation, suppression, etc.).

Identify Missing Values

```
print(df.isna().sum())
```

```
Item_Weight                1463
Item_Fat_Content               0
Item_Visibility                0
Item_Type                      0
Item_MRP                       0
Outlet_Identifier              0
Outlet_Establishment_Year      0
Outlet_Size                 2410
Outlet_Location_Type           0
Outlet_Type                    0
Item_Outlet_Sales              0
dtype: int64
```

Gestion des valeurs incohérentes :

Analyse:

La colonne Item_Fat_Content présente des valeurs incohérentes (LF,reg,low fat).

Identify Inconsistent Values

```
print("--------")
print(df['Item_Fat_Content'].value_counts())
print("--------")
print(df['Item_Type'].value_counts())
print("--------")
print(df['Outlet_Size'].value_counts())
print("--------")
print(df['Outlet_Location_Type'].value_counts())
print("--------")
print(df['Outlet_Type'].value_counts())
print("--------")
```

```
--------
Item_Fat_Content
Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: count, dtype: int64
--------
```

Traitement des Valeurs Manquantes

Item_Weight

Pour Item_Weight, nous avons utilisé la moyenne des poids pour remplacer les valeurs manquantes, assurant la cohérence des données et conservant leur richesse.

```
average_weight = df['Item_Weight'].mean()
df.loc[:, 'Item_Weight'].fillna(average_weight, inplace = True)
```

Outlet_Size

Pour Outlet_Size, les valeurs manquantes ont été remplacées par "Unknown", notamment pour OUT013, OUT017 et OUT045, où la taille est indéterminée. Cela permet de maintenir l'intégrité de l'analyse sans perdre d'information.

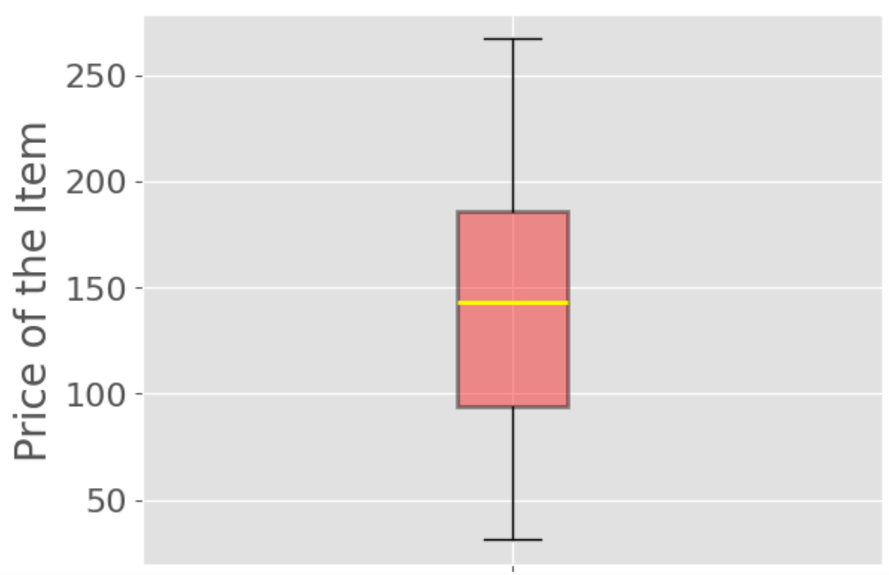
```
df.loc[:, 'Outlet_Size'].fillna("Unknown", inplace = True)
```

Correction des Valeurs Inconsistantes

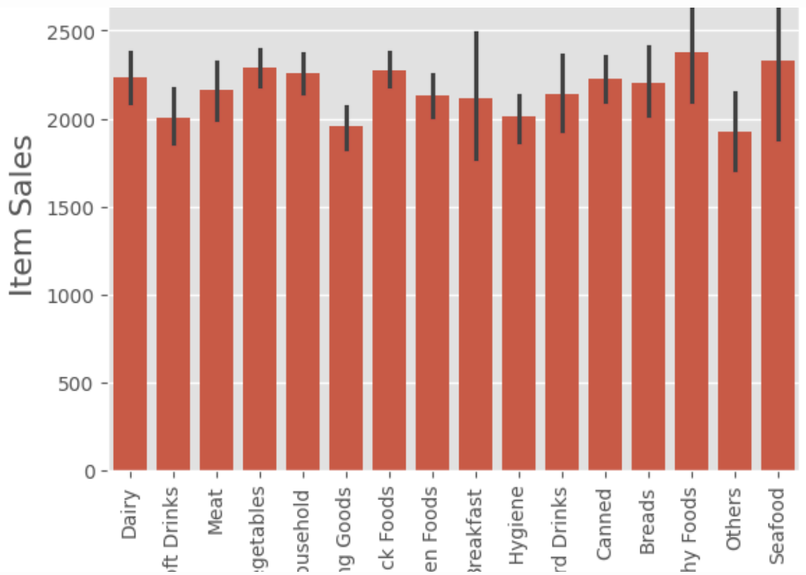
Pour la colonne Item_Fat_Content, nous avons corrigé les valeurs inconsistantes en remplaçant les variations de "Low Fat" et "Regular" par des valeurs standardisées :

- **'LF' et 'low fat' ont été remplacés par 'Low Fat'.**
- **'reg' a été remplacé par 'Regular'.**

```
df['Item_Fat_Content'].replace('LF', 'Low Fat', inplace = True)
df['Item_Fat_Content'].replace('low fat', 'Low Fat', inplace = True)
df['Item_Fat_Content'].replace('reg', 'Regular', inplace = True)
```



Average item price



Comparison of categorical sales

Item Price compared to Item Sales

- **En analysant le prix moyen par article, nous pouvons constater que les articles ne dépassent généralement pas la barre des 250 $. Connaître le prix de nos articles et leur impact sur nos ventes est essentiel pour pouvoir projeter les ventes de manière précise par la suite.**
- **Il n'y a aucune catégorie qui surperforme de manière significative par rapport aux autres.**
- **En général, plus le prix d’un article est élevé, plus les revenus générés par les ventes sont importants.**

6. PRÉPARATION DES DONNÉES

Encodage des Variables Catégoriques

L'encodage est réalisé avec OneHotEncoder, qui transforme les colonnes catégoriques en valeurs numériques.

```
OneHotEncoder(handle_unknown='ignore', sparse_output=False)
```

Ceci fait partie du categorical_pipeline créé avec make_pipeline et appliqué dans le preprocessor via le transformateur de colonnes.

```
categorical_pipeline = make_pipeline(freq_imputer, ohe)
```

Standardisation

Utilisation de StandardScaler pour normaliser les variables numériques.

```
scaler = StandardScaler()
```

Cette transformation fait partie du numeric_pipeline, qui normalise les données numériques dans le preprocessor.

```
numeric_pipeline = make_pipeline(mean_imputer, scaler)
```

Division du Dataset

Division Train-Test : Le code utilise train_test_split() pour diviser le jeu de données en ensembles d'entraînement et de test.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

7. MODÉLISATION

Type de modèle sélectionné : Régression linéaire et Arbre de décision régressif.

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

Justification du Choix du Modèle

- Régression linéaire : Simple, efficace pour les relations linéaires.
- Arbre de décision : Flexible et interprétable, capture les relations non linéaires.

7. MODÉLISATION

Implémentation du Modèle

Code d'entraînement des modèles :

```
lr = LinearRegression()
lr.fit(X_train_processed, y_train);
```

Entraînement de la régression linéaire

```
dec_tree = DecisionTreeRegressor(random_state = 42)
dec_tree.fit(X_train_processed, y_train)
```

Entraînement de l'arbre de décision

8. ENTRAÎNEMENT DU MODÈLE

Paramètres d'Entraînement

- **Régression linéaire : Aucun hyperparamètre spécifique.**

- **Arbre de décision :**
  - **-max_depth : profondeur maximale.**
  - **-random_state : pour reproductibilité.**

```
dec_tree = DecisionTreeRegressor(max_depth = depth,
                                 random_state = 42)
```

8. ENTRAÎNEMENT DU MODÈLE

Techniques utilisées :

- **Validation croisée : Pour une évaluation robuste.**
- **Régularisation : Limitation de max_depth et min_samples_split pour éviter le surapprentissage.**

```
scores = cross_val_score(dec_tree, X_train_processed, y_train, cv=5)
```

9. ÉVALUATION DU MODÈLE

Métriques d'Évaluation :

- **R^2 : Proportion de la variance expliquée.**
- **MSE et RMSE : Moyenne des erreurs quadratiques et leur racine.**

```
dec_tree_train_score = dec_tree.score(X_train_processed, y_train)
dec_tree_test_score = dec_tree.score(X_test_processed, y_test)
```

R^2

```
rmse_train = np.sqrt(mean_squared_error(y_train, train_preds))
rmse_test = np.sqrt(mean_squared_error(y_test, test_preds))
```

RMSE

9. ÉVALUATION DU MODÈLE

1. Régression Linéaire

Scores R^2 :

```
Linear Regression - Train R^2: 0.56, Test R^2: 0.57
```

- **Entraînement : 0.56**
- **Test : 0.57**

RMSE :

```
Linear Regression Train RMSE Score: 1141.7595072117326
Linear Regression Test RMSE Score: 1093.7428273362657
```

- **Entraînement : 1141.76**
- **Test : 1093.74**

9. ÉVALUATION DU MODÈLE

2. Arbre de Décision

Scores R^2 :

```
Decision Tree R^2 Train Score: 0.6039397477322956
Decision Tree R^2 Test Score: 0.5947099753159973
```

- **Entraînement : 0.60**
- **Test : 0.59**

RMSE :

```
Decision Tree Train RMSE Score: 1082.6461900869947
Decision Tree Test RMSE Score: 1057.4431299496732
```

- **Entraînement : 1082.65**
- **Test : 1057.44**

9. ÉVALUATION DU MODÈLE

Interprétation :

- La régression linéaire a des scores R^2 légèrement plus faibles que l'arbre de décision.
- L'arbre de décision montre des erreurs RMSE plus faibles, indiquant une meilleure performance sur les données d'entraînement et de test par rapport à la régression linéaire.

Conclusion :

- L'Arbre de Décision offre de meilleures performances en termes de R^2 et de RMSE par rapport à la régression linéaire. Cependant, les deux modèles pourraient être améliorés par des techniques de réglage de paramètres, des transformations de données ou l'utilisation de modèles plus complexes.

10. AMÉLIORATION DU MODÈLE

Ajustement des Hyperparamètres

Optimisation des hyperparamètres : Le processus consiste à tester différentes valeurs pour les hyperparamètres afin d'optimiser les performances du modèle. Par exemple, pour l'arbre de décision, la profondeur maximale (max_depth) peut être modifiée pour voir comment elle affecte la performance :

```
max_depth_range = list(range(1, 45))
r2 = []
for depth in max_depth_range:
    dec_tree = DecisionTreeRegressor(max_depth = depth,
                                     random_state = 42)
    dec_tree.fit(X_train_processed, y_train)
    score = dec_tree.score(X_test_processed, y_test)
    r2.append(score)
```

THANK YOU

For Your Attention