L'objectif de ce projet est de prédire, via l'apprentissage automatique, le prix de vente d'une maison ou d'un appartement en fonction de sa ville, sa surface, son diagnistique de performance énergétique, etc.

Pour ce faire, on s'appuiera sur le site `https://www.immo-entre-particuliers.com/`, qui a l'avantage de ne pas posséder de protection contre les bots. Par respect pour les propriétaires du site, on veillera à limiter au maximum le nombre de requêtes. En particulier, on s'assurera d'avoir un code fonctionnel avant de scraper l'intégralité des annonces, pour éviter les répétitions.

Instructions

Ce projet est à réaliser impérativement en binôme : les projets individuels seront pénalisés.
Seules les bibliothèques spécifiquement mentionnées dans le sujet sont autorisées.
Le code pour le troisième (et dernier) jalon devra être uploadé sur Eprel au plus tard le **30 mars 2025**.

Premier jalon : récupération des données en `python`

Tout projet de machine learning s'appuie sur un ensemble massif de données. L'objectif de cette première partie est la récupération de ces données, et leur stockage dans le format CSV.
Cette partie est à coder en `python`.

Étudier la page d'une maison ou d'un appartement à vendre sur le site
`https://www.immo-entre-particuliers.com/`.
Une telle page est donnée en exemple dans les Figures 1 et 2.

Question 1 En utilisant les bibliothèques `requests` et `Beautiful soup` comme on l'a vu dans le TP2, écrivez une fonction `getsoup()` qui prend en entrée l'URL d'une annonce, et renvoie la soupe correspondant à cette page HTML.

Question 2 Dans la suite, on va éliminer un certain nombre d'annonces qui ne correspondent pas à certains critères (par exemple les parkings, fonds de commerce, ou les annonces trop peu chères, qui correspondent souvent à une proposition d'échange plutôt qu'à une vente).

Pour ce faire, les fonctions chargées d'extraire les informations importantes de la soupe lèveront une exception `NonValide`.

Créez donc une classe `NonValide`, héritant de la classe `Exception`.

Question 3 En haut de chaque page (cf. l'encadré rouge sur la Figure 1) est indiqué le prix de l'annonce.

Écrivez une fonction `prix()` qui prend en entrée la soupe d'une annonce, et renvoie son prix, sous forme de chaine de caractères et sans le symbole `"€"`. On ne considérera pas les annonces de moins de 10.000€ (qui sont probablement des propositions d'échanges de maison ou d'appartement) : dans ce cas, `prix()` devra lever une exception `NonValide`.

Question 4 Écrivez une fonction `ville()` qui attend en entrée la soupe d'une annonce et renvoie la ville dans laquelle se trouve le bien. Cette information est disponible en haut de la page, comme indiqué en bleu sur la Figure 1. Plus précisément, la ville est située après la dernière occurrence de la sous-chaîne de caractères ", ". On pourra utiliser la fonction `rfind()` de python.

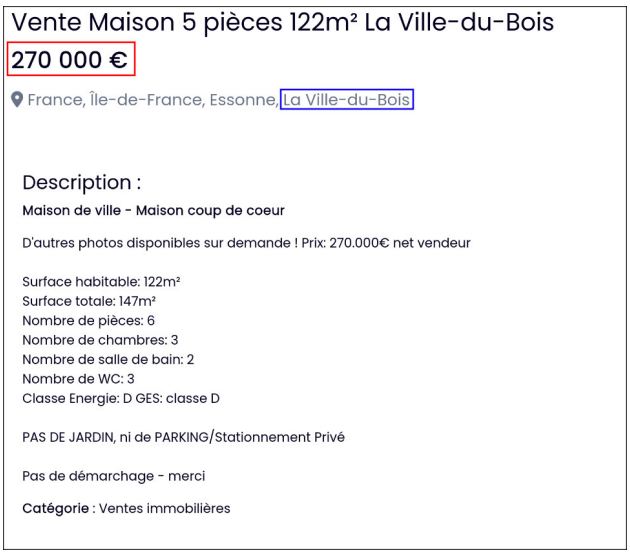

FIGURE 1 – Le prix (en rouge) et la ville (en bleu) de l'annonce.

On va maintenant se pencher sur les autres caractéristiques des annonces. Parmi toutes celles répertoriées sur le site, nous en retiendrons 6, en bleu sur la Figure 2 : le type de bien, sa surface, le nombre de pièces, de chambres, de salles de bain, et le DPE (diagnostique de performance énergétique). Ces informations sont indiquées sous le titre "Caractéristiques" (en rouge sur la Figure 2).

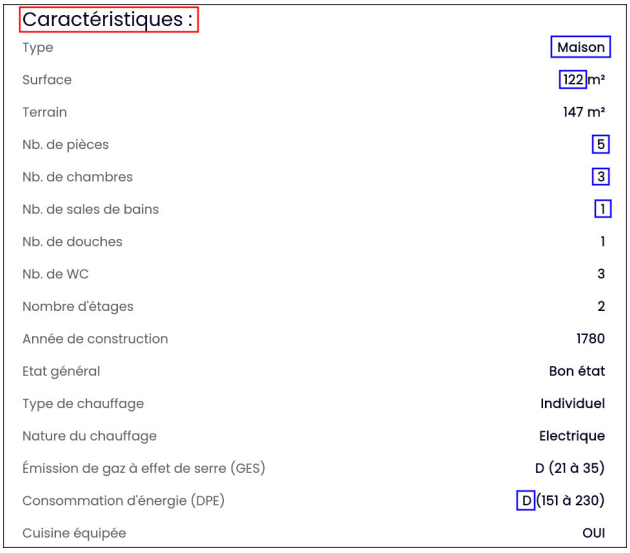

FIGURE 2 – Les caractéristiques de l'annonce. Celles qui nous intéressent sont encadrées en bleu.

Question 5 Écrivez des fonctions `type()`, `surface()`, `nbrpieces()`, `nbrchambres()`, `nbrsdb()` et `dpe()`, qui prennent une soupe en argument et renvoient la valeur de la caractéristique associée (sous forme de chaîne de caractères).

On pourra commencer par écrire une fonction renvoyant la balise (sous forme de soupe) contenant toutes les caractéristiques. Pour cela, on pourra chercher le header `"Caractéristiques :"`, comme indiqué en rouge sur la Figure 2.

Dans la mesure où l'on ne cherche que des maisons et des appartements, `type()` lèvera une exception `NonValide` si le type n'est ni `"Maison"` ni `"Appartement"`.

Pour les cinq autres fonctions, si la donnée est manquante, on renverra la chaîne `"-"`. **En particulier, les annonces aux caractéristiques partielles seront acceptées, du moment qu'il s'agit bien de maisons ou d'appartements d'une valeur d'au moins 10.000€.**

Question 6 Écrivez une fonction `informations()` qui attend la soupe d'une annonce en argument, et renvoie une chaîne de caractères contenant toutes les informations de l'annonce, séparées par des virgules, dans l'ordre :

`"Ville,Type,Surface,NbrPieces,NbrChambres,NbrSdb,DPE,Prix"`

Sur l'annonce représentée dans les Figures 1 et 2, cette fonction renverra donc :

`"La Ville-du-Bois,Maison,122,5,3,1,D,270000"`

Cette fonction `informations()` lèvera donc une exception `NonValide` en cas d'annonce non conforme.

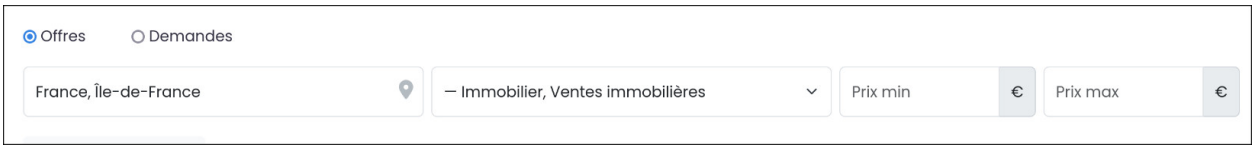

FIGURE 3 – Recherche des offres de ventes immobilières en Île-de-France.

Question 7 Nous allons limiter nos recherches aux annonces de ventes immobilières dans la région Île-de-France. On veillera à se limiter aux offres (cf. Figure 3).

En étudiant l'URL et la structure des différentes pages de résultats (il devrait y en avoir autour d'une centaine), écrivez un script qui parcourt toutes les annonces proposées par le site et appelle `informations()` sur les soupes correspondantes (en attrapant les exceptions `NonValide` soulevées).

Les résultats obtenus devront être enregistrés, à raison d'une ligne par annonce, dans un fichier CSV (comma-separated values) dont la première ligne indiquera les étiquettes (sans guillemets) des différents champs :

```
Ville,Type,Surface,NbrPieces,NbrChambres,NbrSdb,DPE,Prix
```

Pour la suite du projet, on travaillera à partir de ce fichier CSV local.

À titre indicatif, il y avait 449 annonces valides (et donc 450 lignes au total dans le fichier CSV) au jour où nous rédigeons cet énoncé. Ce nombre est évidemment susceptible d'évoluer en fonction des annonces ajoutées ou retirées du site, mais les variations sont probablement minimes. Si vous déviez beaucoup de ce nombre, c'est que votre code est sans doute incorrect.

Deuxième jalon : Nettoyage des données

Avant de pouvoir utiliser notre jeu de données dans le cadre d'un apprentissage automatique, il va nous falloir les analyser et les nettoyer. Pour cela, on utilisera la bibliothèque `pandas`.

Question 8 Commencez par importer le contenu du CSV de la partie précédente dans un `DataFrame` nommé `annonces`, et vérifiez la conformité des données.

La bibliothèque d'apprentissage automatique `scikit-learn`, que nous utiliserons dans la partie suivante, ne peut manipuler que des données numériques. Il va falloir procéder à un certain nombre de conversions et de complétions.

Valeurs manquantes

Certains champs d'`annonces` sont vides. Nous allons les remplir d'une manière adaptée au contexte.

Question 9 À l'aide de la méthode `replace()` de `DataFrame`, modifiez `annonces` en remplaçant toutes les valeurs manquantes (symbolisées par la chaîne de caractères `"-"`) de `DPE` par la valeur `"Vierge"`.

Question 10 Modifiez `annonces` en remplaçant le valeurs manquantes de chacune des colonnes `Surface`, `NbrPieces`, `NbrChambres` et `NbrSdb` par la valeur moyenne de la colonne en question.

Pour pouvoir calculer la moyenne, il faudra faire attention au type des colonnes : ces types pourront être modifiés grâce à la méthode `astype()`.

Vérifiez qu'à ce stade, plus aucune ligne ne contienne de valeur manquante. S'il y en a, on les supprimera de la `DataFrame`.

Colonnes `"Type"` et `"DPE"`

On va maintenant se concentrer sur les deux colonnes `"Type"` et `"DPE"`. Pour ces colonnes, on va utiliser la méthode des variables indicatrices. Cela consiste, par exemple pour la colonne `"DPE"`, à créer une nouvelle colonne pour chaque valeur `v` possible. Une ligne possèdera un `1` dans cette colonne si et seulement si son `DPE` vaut `v`. Un exemple minimal est donné en Figure 4.

	DPE	···
0	Vierge	···
1	A	···
2	B	···
3	Vierge	···

(a) Avant

	DPE_A	**DPE_B**	**DPE_Vierge**	···
0	0	0	1	···
1	1	0	0	···
2	0	1	0	···
3	0	0	1	···

(b) Après

FIGURE 4 – Illustration de l'introduction de variables indicatrices

Question 11 En utilisant la fonction `get_dummies()` de `pandas`, créez des variables indicatrices pour les colonnes `"DPE"` et `"Type"`.

Colonne `"Ville"`

Plutôt que d'introduire des variables indicatrices pour la colonne `"Ville"` (il y en aurait plusieurs centaines), on va remplacer chaque ville par sa latitude et sa longitude. En plus d'obtenir des valeurs numériques, cette manière de faire nous permet de profiter du fait que deux villes géographiquement proches connaissent probablement des prix de l'immobilier assez similaires. Ainsi, nos modèles pourront raisonnablement espérer estimer le prix d'un bien situé dans une ville qui n'est pas présente dans l'ensemble d'entraînement.

Pour connaître les latitude et longitude de chaque ville française, nous allons travailler à partir des données mises à disposition par le gouvernement au format CSV.

Question 12 Téléchargez le fichier `cities.csv` et importez-en les données dans une `DataFrame` nommée `villes`.

Avant de faire la jointure en les deux tables, il faut nous assurer que le nom des villes soit dans le même format de part et d'autre.

Question 13 Modifiez les valeurs de la colonne `"label"` de `villes` et de la colonne `"Ville"` de `annonces` pour que les villes soient nommées de la même manière. On pourra par exemple décider de supprimer tous les espaces, tirets, apostrophes, mettre les noms en minuscules, enlever les accents... Il faudra étudier les lignes sur lesquelles la jointure est impossible pour gérer les cas particuliers. On fera en particulier attention au cas de Paris et ses arrondissements.

On pourra pour cela utiliser les méthodes `Series.str.replace()` et `Series.str.lower()`.

Question 14 En utilisant la fonction `merge()` (qui permet de faire une jointure entre deux tables), complétez `annonces` avec deux colonnes `"latitude"` et `"longitude"`.

Supprimez ensuite les colonnes `"Ville"` et `"label"`, dont nous n'avons plus besoin.

À ce stade de l'énoncé, `annonces` devrait contenir 16 colonnes, toutes numériques et sans valeurs manquantes.

Troisième jalon : Apprentissage

Nous sommes prêts à passer à l'apprentissage à proprement parler. Pour ce faire, nous allons utiliser la bibliothèque `scikit-learn`.

Préparation de données d'apprentissage et de test

Le but est d'estimer le prix d'un bien à partir des attributs. La colonne `"Prix"` correspond ainsi aux étiquettes ("targets"), tandis que les 15 autres colonnes sont les attributs ("features").

Question 15 Préparez la matrice de données et le vecteur des étiquettes pour votre jeu de données. Séparez les données en un jeu d'entraînement et un jeu de test, sur un base 75%/25%, en utilisant le paramètre `random_state = 49`.

Nous utiliserons les notations suivantes :

- `X_train` (resp. `y_train`) fait référence aux données d'apprentissage (resp. à leur étiquette),
- `X_test` (resp. `y_test`) fait référence aux données de test (resp. à leur étiquette).

Premier modèle : Régression Linéaire

Dans un premier temps, nous allons appliquer une *régression linéaire* `LinearRegression` sur nos données.

Question 16 Évaluez ce modèle sur le jeu de test par r^2`_score`. Rappelons qu'il s'agit de la mesure par défaut pour la fonction `score()` des méthodes de régression.

Question 17 Nous allons maintenant évaluer l'impact du pré-traitement sur nos données. Afin de comparer les résultats, nous allons construire un tableau en précisant la méthode appliquée et le score obtenu.

En utilisant la commande `make_pipeline`, normalisez d'abord les données avant de lancer l'apprentissage, et évaluez le score sur le jeu de test. Répétez l'expérience, cette fois-ci en standardisant les données avant l'apprentissage.

Question 18 Remplissez le tableau suivant avec les résultats obtenus grâce à la régression linéaire, avec ou sans pré-traitement. Constatez-vous une amélioration des prédictions due au pré-traitement pour ce jeu de données et cette méthode ?

Méthode	r^2
LR	
Normalisation + LR	
Standardisation + LR	

Deuxième modèle : Arbre de Décision

Le deuxième modèle d'apprentissage que nous allons considérer sont les *arbres de décision* `DecisionTreeRegressor`.

Question 19 Évaluez ce modèle sur le jeu de test, en employant dans une premier temps le paramètre `max_depth = 4`. Testez ensuite l'impact du pré-traitement, comme pour la régression linéaire.

Constatez-vous une amélioration significative suite au pré-traitement ? Les scores obtenus sont-ils satisfaisants ?

Incrementez ensuite le paramètre `max_depth`. L'augmentation de la profondeur permet-elle de meilleurs résultats ? Si cette amélioration n'est pas significative, nous conserverons `max_depth = 4` pour la suite, par souci de simplicité.

Question 20 Remplissez à nouveau un tableau de trois lignes (une pour la méthode "AD", une pour "Normalisation + AD" et une pour "Standardisation + AD") avec les résultats que vous avez obtenus grâce aux arbres de décision, avec ou sans pré-traitement.

Troisième modèle : N plus proches voisins

En guise de troisième méthode d'apprentissage, nous allons nous pencher sur la méthode des *plus proches voisins* `KNeighborsRegressor`.

Question 21 Évaluez ce modèle sur le jeu de test, en employant dans une premier temps le paramètre `n_neighbors = 4`. Testez ensuite l'impact du pré-traitement, comme pour les modèles précédents.

Constatez-vous une amélioration significative suite au pré-traitement ? Les scores obtenus sont-ils satisfaisants ?

Refaites l'expérience avec un paramètre `n_neigbors=5`. L'augmentation du nombre de voisins considérés permet-elle de meilleurs résultats ? Si cette amélioration n'est pas significative, nous conserverons `n_neighbors = 4` pour la suite, par souci de simplicité.

Question 22 Remplissez à nouveau un tableau de trois lignes (une pour la méthode "KNN", une pour "Normalisation + KNN" et une pour "Standardisation + KNN") avec les résultats que vous avez obtenus grâce à la méthode des plus proches voisins, avec ou sans pré-traitement.

Discussions sur le jeu de données

Nous avons considéré à ce stade trois méthodes d'apprentissage, à savoir la régression linéaire (LR), les arbres de décision (AD) et la méthode des plus proches voisins (KNN). En gardant uniquement le meilleur score obtenu pour chacune de ces trois méthodes (avec pré-traitement ou non), nous obtenons un tableau de 3 lignes :

Méthode	r^2
LR	
AD	
KNN	

Question 23 Replissez le tableau, et comparez les scores obtenus pour ce jeu de données grâce à ces différentes méthodes. Que constatez-vous ? Dans la suite, nous noterons $\mathcal{M}$ la méthode qui donne le meilleur score pour les données de test.

Visualisation des résultats de test

Nous cherchons maintenant à visualiser les résultats obtenus grâce à la méthode $\mathcal{M}$.

À titre d'exemple, la Figure 5 représente pour les résultats que nous avons obtenus – les vôtres seront probablement légèrement différents. Dans ce graphique, l'axe des x représente les étiquettes `y_test` tandis que sur l'axe des y figurent les estimations obtenues par la méthode $\mathcal{M}$. Les points sont donc les estimations faites par le modèle pour les données de test. Si les points sont sur la diagonale `estimation=y_test`, c'est que notre modèle d'apprentissage est "parfait". À l'inverse, les points éloignés de la diagonale sont là où les erreurs d'estimation sont les plus importantes.

Question 24 Dessinez ce graphique pour représenter vos résultats. Que constatez-vous ? Pensez-vous que le jeu de données est suffisant pour construire un modèle d'apprentissage précis ?

Réduction le nombre d'attributs

Nous souhaitons maintenant réduire le nombre d'attributs de notre jeu de données en utilisant la méthode `PCA`.

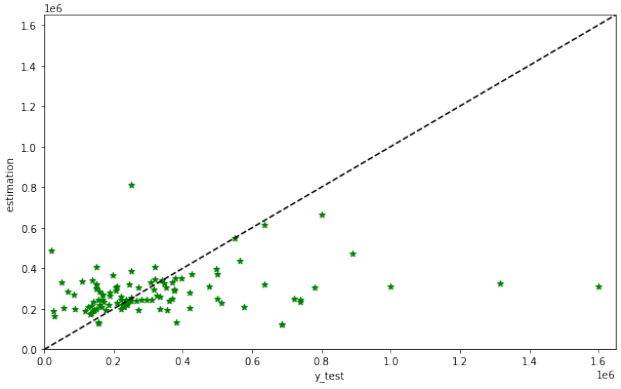

FIGURE 5 – `y_test` versus estimation

Question 25 Réduisez le nombre de composantes à 2. La modélisation avec 2 composantes principales est-elle satisfaisante pour ce jeu de données ? Justifiez.

Question 26 Lancez la méthode $\mathcal{M}$ sur les données obtenues après réduction des composantes, et comparez les nouveaux scores aux scores précédents. Que constatez-vous ?

Matrice de corrélation

Pour terminer, nous allons tenter de comprendre dans quelle mesure chaque attribut permet de prédire l'étiquette.

Question 27 Établissez la matrice de corrélation `DataFrame.corr()` du jeu de données. On pourra trouver un exemple de création et d'affichage de matrice de corrélation à cette adresse.

Question 28 Quel est l'attribut dont la connaissance nous renseigne le plus sur le prix du bien immobilier ?

Pour déterminer cela, on regardera la colonne (ou la ligne) "Prix" de la matrice, qui indique la corrélation entre le prix et chacun des autres attributs, considérés individuellement. Plus le résultat est grand, plus la corrélation entre le prix et cet attribut est grande. À l'inverse, plus ce résultat est grand dans les négatifs, plus la corrélation inverse est grande. Enfin, un résultat proche de zéro signifie qu'il y a une faible corrélation entre le prix et cet attribut.

Question 29 Modifiez le jeu de données en ne conservant que les cinq attributs qui sont le plus corrélés (positivement ou négativement) au prix. Appliquez la méthode $\mathcal{M}$ à ce nouveau jeu de données, et comparez le score de prédiction aux résultats obtenus précédemment. Que constatez-vous ?