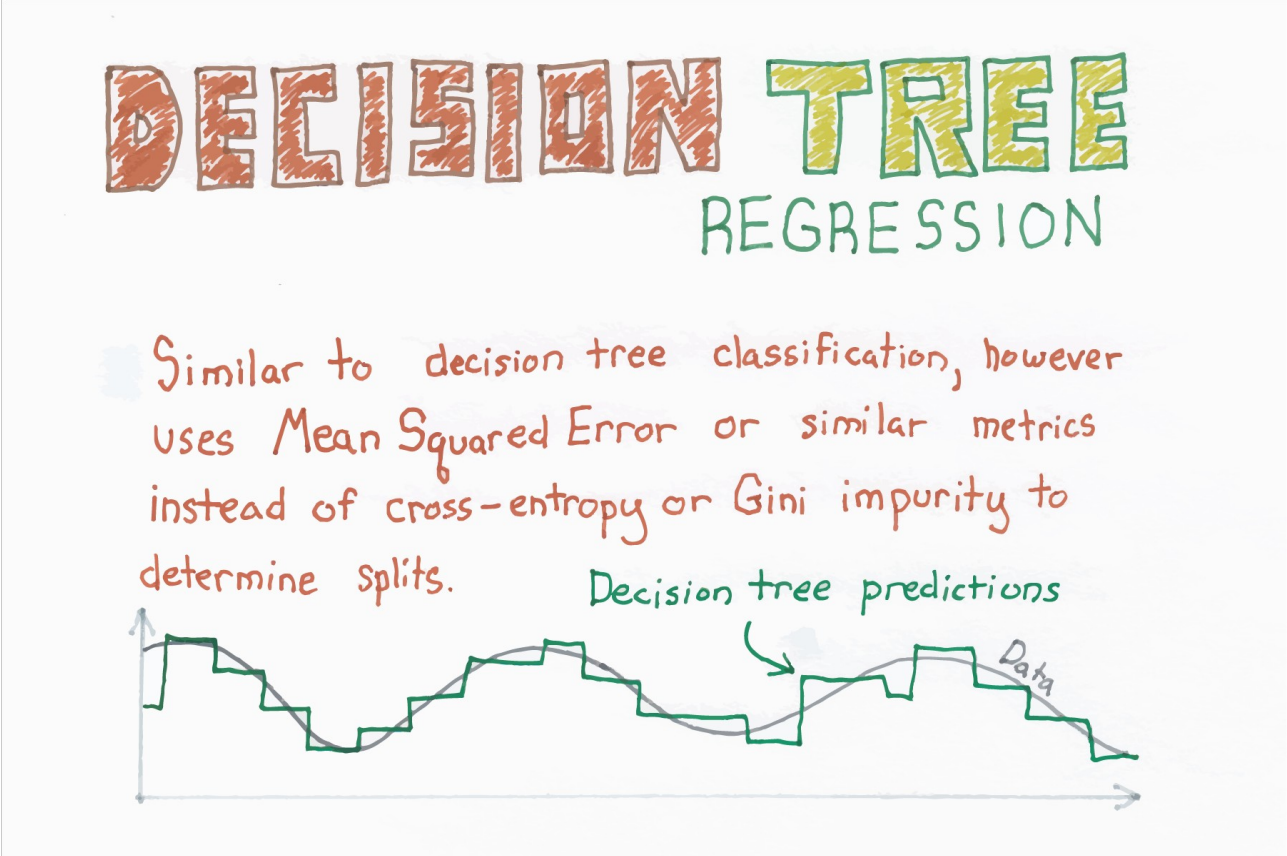
Apprentissage supervisé - Regression, DecisionTreeRegressor, RandomForestRegressor



Sommaire

Objectifs du Brief....2
1 Préparation des données....2
1.1 Téléchargement de données....2
1.2 Information sur les données....3
1.3 Répartition des données....3
1.4 Découverte et visualisation des données....5
....5
1.5 Nettoyage des données....7
2 Sélection, apprentissage et évaluation du modèle....8
2.1 Régression Linéaire....8
2.1.1 Entraînement du modèle....8
2.1.2 Évaluation du jeu d'entraînement....8
2.2 L'arbre de décision....8
2.2.1 Entraînement du modèle....8
2.2.2 Évaluation du jeu d'entraînement....9
2.3 Validation Croisée....9
2.3.1 Validation croisée sur le modèle de l'arbre de décision....9
2.3.2 Validation croisée sur le modèle de la régression linéaire....9
3 Fine-Tunning....10
3.1 Grid Search....10
3.2 Evaluation sur la base de test....10
Conclusion....10

Objectifs du Brief

- Réaliser un projet individuel de la préparation de données, d'apprentissage automatique à la phase de prédiction
- L'objectif de ce projet d'apprentissage automatique est de pouvoir, à partir de plusieurs caractéristiques (features), d'estimer les prix des maisons (target) dans l'état de la californie.
- L'interprétation des résultats
- La prise en main des bibliothèques (Pandas, Matplotlib, Scikit-learn, Numpy)

1 Préparation des données

1.1 Téléchargement de données

img = cv2.imread('data/winter-5814578_960_720.jpg',1)

```
----------> Jeu de données
```

Out[25]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY
...	...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845.0	330.0	1.5603	78100.0	INLAND
20636	-121.21	39.49	18.0	697.0	150.0	356.0	114.0	2.5568	77100.0	INLAND
20637	-121.22	39.43	17.0	2254.0	485.0	1007.0	433.0	1.7000	92300.0	INLAND
20638	-121.32	39.43	18.0	1860.0	409.0	741.0	349.0	1.8672	84700.0	INLAND
20639	-121.24	39.37	16.0	2785.0	616.0	1387.0	530.0	2.3886	89400.0	INLAND

20640 rows × 10 columns

1.2 Information sur les données

Entrée [28]:
```
# Code qui affiche le nombre de lignes et de colonnes des données, le type des at- tributs et le nombre de valeurs non nulles.
print("--------> Infos sur le fichier , attributs, types.... \n")
print(data1.info())
```

```
--------> Infos sur le fichier , attributs, types....

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20640 non-null  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
None
```

On remarque tout de suite qu'il manque des données à la colonne « total_bedrooms » : 20433 features contre 20640 pour les autres colonnes.

De plus , on voit aussi que « ocean_proximity » possède des valeurs qualitatives.

1.3 Répartition des données

On détermine la target , qui est ici "median_house_value" puis on partitionne les données en base d'apprentissage et en base de test.

from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size= 0.2, random_state=42)

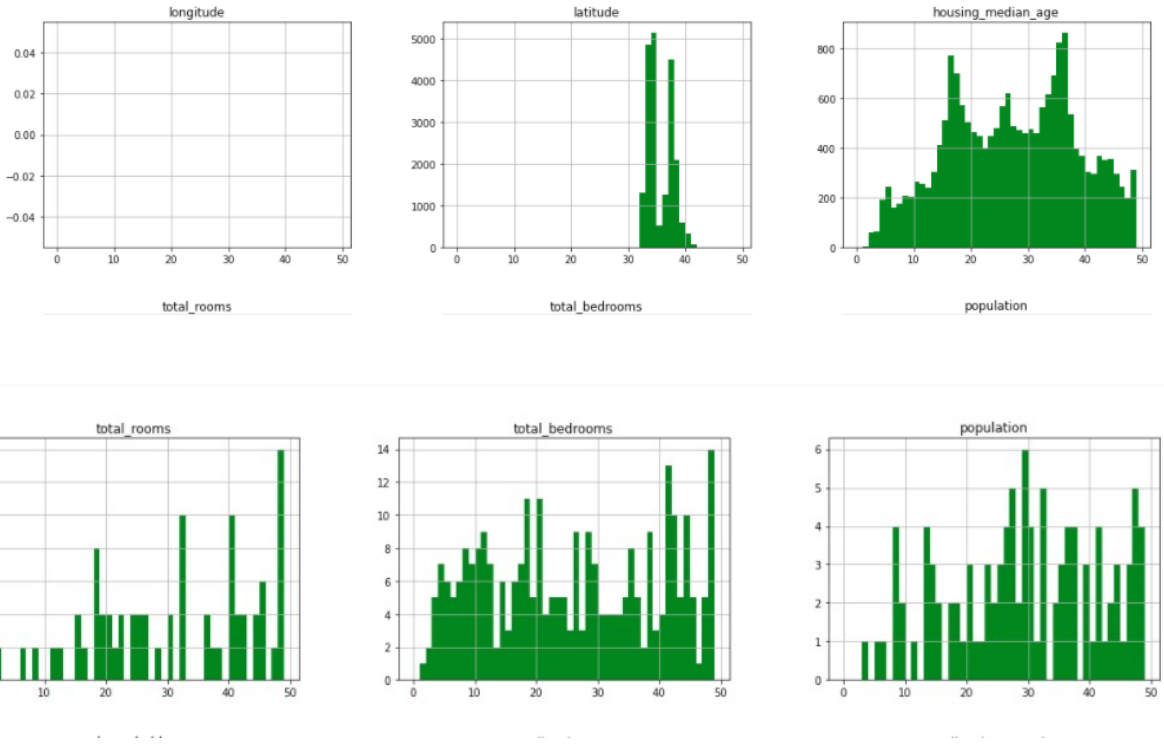
Entrée [31]:
```
# Détermination de la Target et des prédicables
X = data1.iloc[:, 0:9]
Y = data1["median_house_value"]
print("-----> Affichage de X ")
print(X)
```

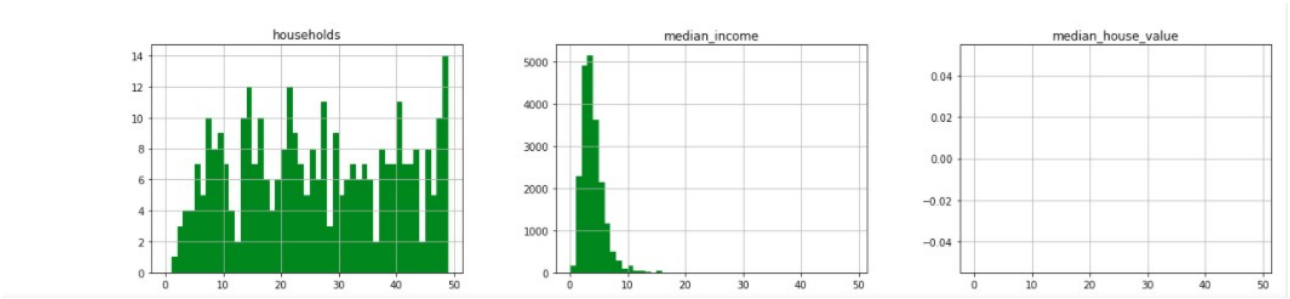
```
-----> Affichage de X
       longitude  latitude  housing_median_age  total_rooms  total_bedrooms  \
0        -122.23     37.88                41.0        880.0           129.0
1        -122.22     37.86                21.0       7099.0          1106.0
2        -122.24     37.85                52.0       1467.0           190.0
3        -122.25     37.85                52.0       1274.0           235.0
4        -122.25     37.85                52.0       1627.0           280.0
...          ...       ...                 ...          ...             ...
20635    -121.09     39.48                25.0       1665.0           374.0
20636    -121.21     39.49                18.0        697.0           150.0
20637    -121.22     39.43                17.0       2254.0           485.0
20638    -121.32     39.43                18.0       1860.0           409.0
20639    -121.24     39.37                16.0       2785.0           616.0

       population  households  median_income  median_house_value
0           322.0       126.0         8.3252            452600.0
1          2401.0      1138.0         8.3014            358500.0
2           496.0       177.0         7.2574            352100.0
3           558.0       219.0         5.6431            341300.0
4           565.0       259.0         3.8462            342200.0
...           ...         ...            ...                 ...
20635       845.0       330.0         1.5603             78100.0
20636       356.0       114.0         2.5568             77100.0
20637      1007.0       433.0         1.7000             92300.0
20638       741.0       349.0         1.8672             84700.0
20639      1387.0       530.0         2.3886             89400.0

[20640 rows x 9 columns]
```

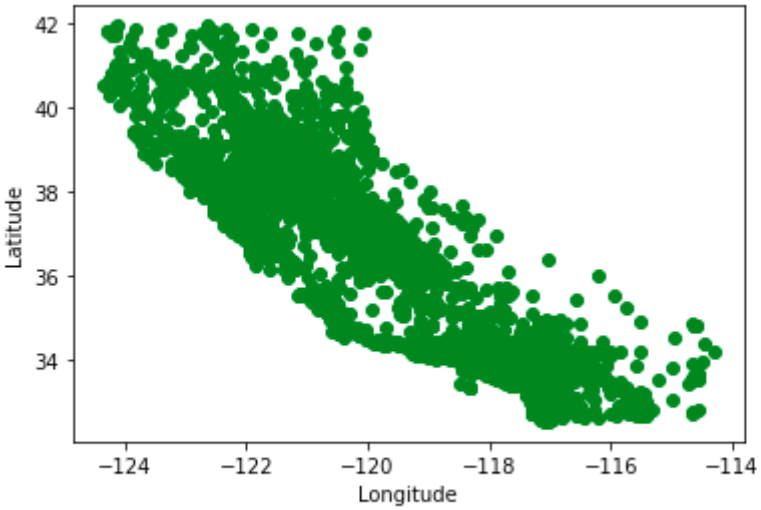
1.4 Découverte et visualisation des données

```
-------> Histogramme des attributs
```

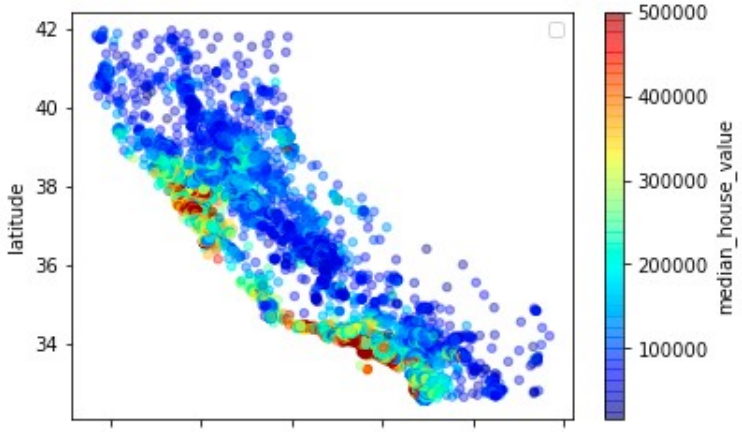
```
Out[32]: array([[<AxesSubplot:title={'center':'longitude'}>,
                 <AxesSubplot:title={'center':'latitude'}>,
                 <AxesSubplot:title={'center':'housing_median_age'}>],
                [<AxesSubplot:title={'center':'total_rooms'}>,
                 <AxesSubplot:title={'center':'total_bedrooms'}>,
                 <AxesSubplot:title={'center':'population'}>],
                [<AxesSubplot:title={'center':'households'}>,
                 <AxesSubplot:title={'center':'median_income'}>,
                 <AxesSubplot:title={'center':'median_house_value'}>]],
               dtype=object)
```




households
median_income
median_house_value


Latitude
Longitude
42
40
38
36
34
−124
−122
−120
−118
−116
−114

```
Out[37]: <matplotlib.legend.Legend at 0x1bbe3432520>
```



1.5 Nettoyage des données

Ce nettoyage a pour objectif de combler les données manquantes et remplacer les donnés qualitatives par des données numériques pour ensuite lancer l'apprentissage correctement .

```
# Code qui remplace les valeurs manquantes par la médiane.
median = data1["total_bedrooms"].median()
data1["total_bedrooms"].fillna(median, inplace=True)
```

```
# Transformation des valeurs qualitatives en des valeurs numériques de la colomne ocean_proximty
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
data2 = data1["ocean_proximity"]
data2_encoded = encoder.fit_transform(data2)
```

2 Sélection, apprentissage et évaluation du modèle

2.1 Régression Linéaire

2.1.1 Entraînement du modèle

On lance le modèle avec les données d'entraînement :

```
from sklearn.linear_model import LinearRegression

linearR = LinearRegression()

linearR.fit(X_train, Y_train)
```

Puis on prédit les classes de la base d'apprentissage :

```
Y_predict = linearR.predict(X_train)
```

2.1.2 Évaluation du jeu d'entraînement

```
# Mesure RMSE du modèle de la régression linéaire

from sklearn.metrics import mean_squared_error

mseL = mean_squared_error(Y_train, Y_predict)
```

```
RMSE Linear Regresion =
4811134397.884195
```

L'erreur de prédiction est trop élevé et doit être proche de 0. Il faut améliorer le modèle.

2.2 L'arbre de décision

2.2.1 Entraînement du modèle

On lance le modèle avec les données d'entraînement :

```
from sklearn.tree import DecisionTreeRegressor

DTregressor = DecisionTreeRegressor()

DTregressor.fit(X_train, Y_train)
```

2.2.2 Évaluation du jeu d'entraînement

mseDTR = mean_squared_error(Y_train, Y_predict)

print('RMSE Decision Tree Regression = ')

print(mseDTR)

RMSE Decision Tree Regression =
4811134397.884195

L'erreur semble la même . Elle est aussi élevé qu'avec la régression linéaire.

2.3 Validation Croisée

2.3.1 Validation croisée sur le modèle de l'arbre de décision

La validation croisée est idéale pour tester l'arbre de décision sur un jeu d'entraînement plus petit.

On lance le modèle :

from sklearn.model_selection import cross_val_score

cvs = cross_val_score(DTregressor, X_train, Y_train,scoring ="neg_mean_squared_error", cv = 10)

```
RMSE Validation Croisée =
[71689.14745536 73026.34926911 67203.18364379 68279.38299113
 70918.06638446 66508.94683068 68863.21082944 70054.20877068
 70207.91540238 71344.93573593]
```

Moyenne des MSE : 69809.53473129666

Ecart type de tous les folds : 1964.9313152145362

2.3.2 Validation croisée sur le modèle de la régression linéaire

On lance le modèle :

from sklearn.model_selection import cross_val_score

cvs = cross_val_score(linearR, X_train, Y_train,scoring ="neg_mean_squared_error", cv = 10)

```
RMSE Validation Croisée =
[66170.13482881 72783.99723185 68950.52987763 67640.0509114
```

```
 70438.49542911 66523.65704676 66541.25492139 70992.53769382
 74292.46725992 70675.67306462]
```

Moyenne des MSE : 69500.87982653153

Ecart type de tous les folds : 2659.0438508863526

La validation croisée sur la régression linéaire ne semble pas beaucoup mieux qu'avec l'arbre de décision . L'écart type est plus élevé même si la moyenne des erreurs baisse.

3 Fine-Tunning

3.1 Grid Search

On crée d'un objet de la classe RandomForestRegressor :

```
from sklearn.ensemble import RandomForestRegressor
RFregressor = RandomForestRegressor()
```

On crée la variable param_grid :

```
param_grid = {'n_estimators':[3, 10, 30], 'max_features': [2, 4, 6, 8]}
```

On entraîne le modèle Grid Search :

```
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(RFregressor, param_grid, cv = 5)
grid_search.fit(X_train, Y_train)
```

On affiche les meilleurs paramètres de la méthode RandomForestRegressor en utilisant la fonction best_params_ :

```
grid_search.best_params_
```

Out : {'max_features': 6, 'n_estimators': 30}

Le max_feature est proche de la valeur maximum 8

3.2 Evaluation sur la base de test

On remplace des valeurs NaN de l'attribut "total_bedrooms" de la base de test par la médiane :

median = data1["total_bedrooms"].median()

X_test["total_bedrooms"].fillna(median, inplace=True)

On transforme des valeurs textuelles de "ocean_proximity" en valeurs numériques :

data_e2 = data1["ocean_proximity"]

data2_encoded = encoder.fit_transform(data_e2)

On stocke le modèle d'apprentissage dans une variable en utilisant la fonction **best_estimator_** du modèle Grid_search

model3 = grid_search.best_estimator_

RandomForestRegressor(max_features=6, n_estimators=30)

Le meilleur estimateur a été trouvé.

Conclusion

Il faut constamment évaluer le modèle pour en tirer le meilleur ajustement en ajoutant de nouvelles données.