Silicon Valley Machine Learning

Récupération du Dataset

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
data = pd.read_csv('train data.csv')
data
```

	Unnamed: 0	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	2072	-119.84	36.77	6.0	1853.0	473.0	1397.0	417.0	1.4817	72000.0	INLAND
1	10600	-117.80	33.68	8.0	2032.0	349.0	862.0	340.0	6.9133	274100.0	<1H OCEAN
2	2494	-120.19	36.60	25.0	875.0	214.0	931.0	214.0	1.5536	58300.0	INLAND
3	4284	-118.32	34.10	31.0	622.0	229.0	597.0	227.0	1.5284	200000.0	<1H OCEAN
4	16541	-121.23	37.79	21.0	1922.0	373.0	1130.0	372.0	4.0815	117900.0	INLAND
...	...	...	...	...	...	...	...	...	...	...	...
16507	1099	-121.90	39.59	20.0	1465.0	278.0	745.0	250.0	3.0625	93800.0	INLAND
16508	18898	-122.25	38.11	49.0	2365.0	504.0	1131.0	458.0	2.6133	103100.0	NEAR BAY
16509	11798	-121.22	38.92	19.0	2531.0	461.0	1206.0	429.0	4.4958	192600.0	INLAND
16510	6637	-118.14	34.16	39.0	2776.0	840.0	2546.0	773.0	2.5750	153500.0	<1H OCEAN
16511	2575	-124.13	40.80	31.0	2152.0	462.0	1259.0	420.0	2.2478	81100.0	NEAR OCEAN

16512 rows × 11 columns

Analyse rapide du Dataset

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16512 entries, 0 to 16511
Data columns (total 11 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Unnamed: 0          16512 non-null  int64  
 1   longitude           16512 non-null  float64
 2   latitude            16512 non-null  float64
 3   housing_median_age  16512 non-null  float64
 4   total_rooms         16512 non-null  float64
 5   total_bedrooms      16336 non-null  float64
 6   population          16512 non-null  float64
 7   households          16512 non-null  float64
 8   median_income       16512 non-null  float64
 9   median_house_value  16512 non-null  float64
 10  ocean_proximity     16512 non-null  object 
dtypes: float64(9), int64(1), object(1)
memory usage: 1.4+ MB
```

```
data['total_bedrooms'].value_counts(dropna = False)
```

```
NaN       176
280.0      46
291.0      41
315.0      41
287.0      40
         ... 
1995.0      1
2190.0      1
1555.0      1
1172.0      1
1183.0      1
Name: total_bedrooms, Length: 1829, dtype: int64
```

```
# we have exactly 176 missing values for "total_bedroom" column
```

```
# vérification des duplicate
data.duplicated().sum()
```

```
0
```

Homogénéisation des données

```python
data["blue_near"] = data.ocean_proximity.apply(lambda x: 1 if x == "<1H OCEAN"
                                               else 2 if x == "INLAND" else
                                               3 if x == "NEAR OCEAN" else
                                               4 if x == "NEAR BAY" else
                                               5 if x == "ISLAND" else
                                               "NaN")
```

```python
data
```

```python
data = data.drop(["Unnamed: 0", "ocean_proximity"], axis = 1)
```

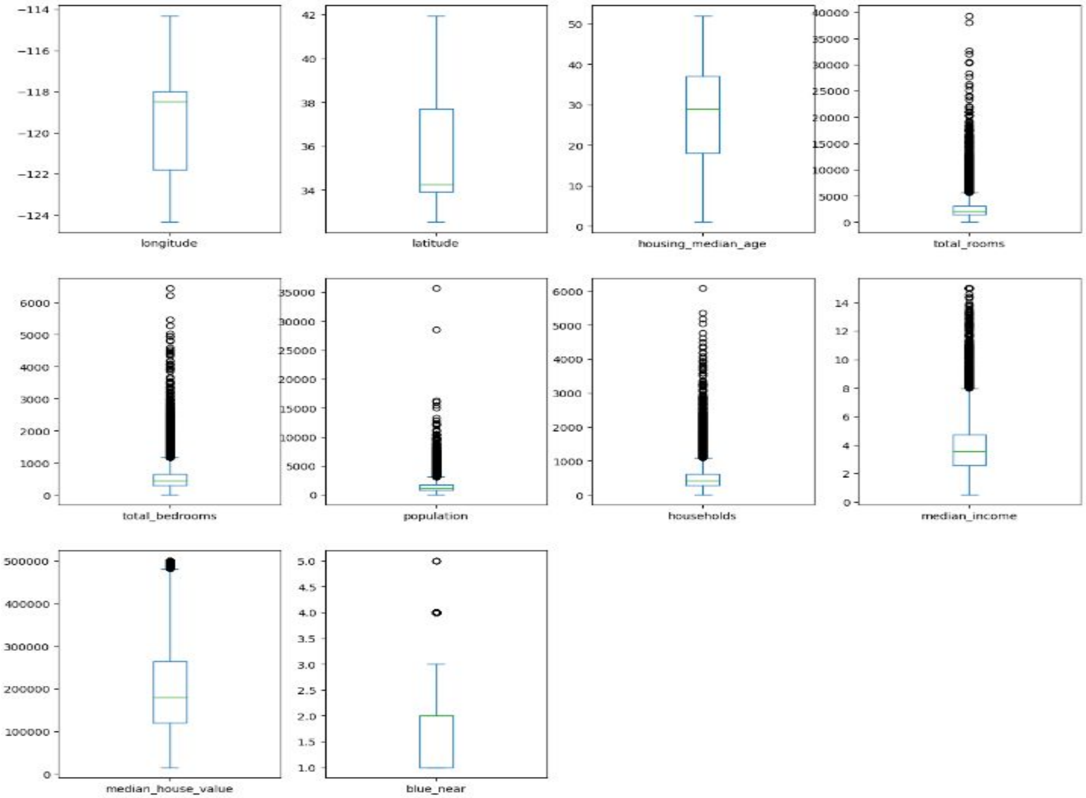
On “standardise” les types de données.

On passe en numérique, des données qui ne le sont pas.

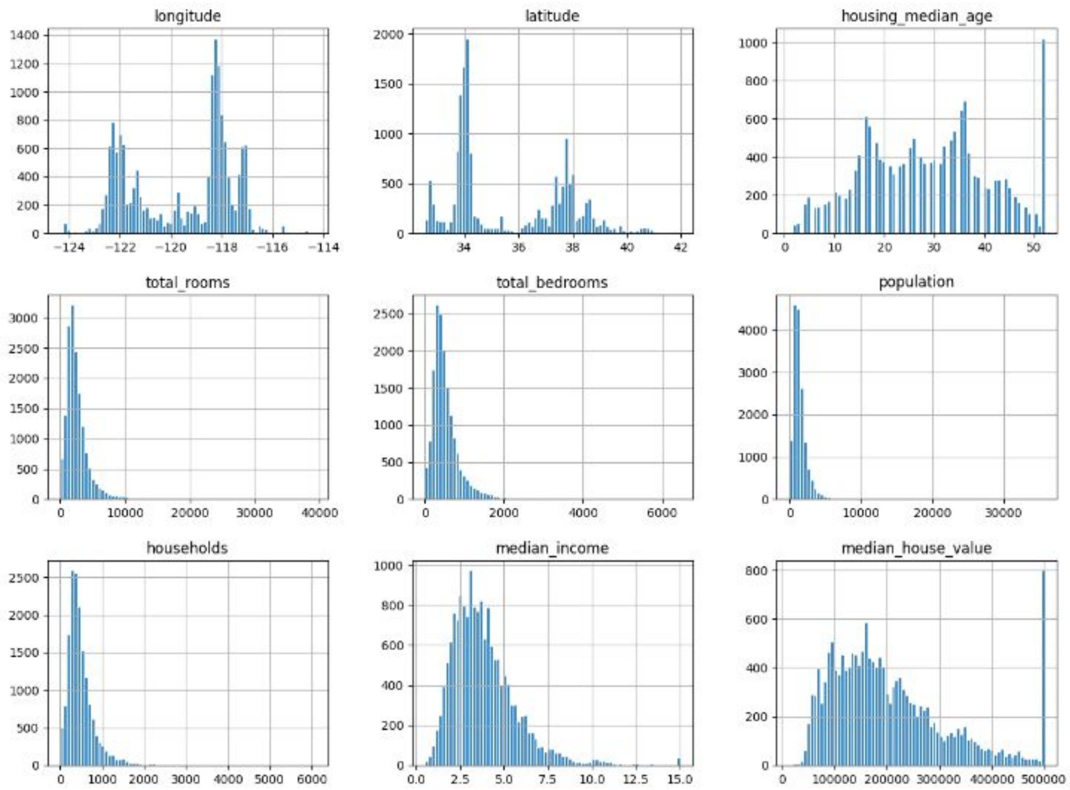
Cela dans le but de ne pas perturber le model et aussi pour pouvoir travailler d’une manière plus

homogène avec les autres features.

Observation des outliers



Observation de la distribution des features



Traitement des NaN et Création du model de Référence

```
imputer = SimpleImputer(strategy='median')
x = imputer.fit_transform(data)
data = pd.DataFrame(x, columns = data.columns, index = data.index)
```

```
data.info()
```

J’ai choisi d’imputer les valeurs manquantes par la moyenne.

```
X = data[['longitude','latitude','housing_median_age','total_rooms','total_bedrooms','population','households','median_income','blue_near']]
y = data['median_house_value']
```

```
# we split the dataset in 70% train, 30% test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 0)
```

```
# now we get the baseline, we will use several and differents aprproach to determine the better prediction algorithm
```

Comparaison de différents modèles

```
def check_model (model, X_train, X_test, y_train, y_test):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_pred,y_test)
    rmse = mean_squared_error(y_test, y_pred)
    score = model.score(X_test, y_test)
    return "mean absolute error is:" f"{mae}", \
           "r2 score is:" f"{r2}", \
           "root mse is:" f"{rmse}", \
           "last but not least, the model score is" f"{score}"
```

```
# model
model = LinearRegression()
# 5 - Fold Cross validate model
cv_results = cross_validate(model, X, y, cv =5)

cv_results['test_score'].mean()
```

```
0.6352559995036742
```

```
model = DecisionTreeRegressor()
check_model (model, X_train, X_test, y_train, y_test)
```

```
('mean absolute error is:46568.23354864756',
 'r2 score is:0.5940701178100725',
 'root mse is:5371731271.947719',
 'last but not least, the model score is0.5903039512303361')
```

```
model = LinearRegression()
check_model (model, X_train, X_test, y_train, y_test)
```

```
('mean absolute error is:50888.819009882776',
 'r2 score is:0.4106975255730O443',
 'root mse is:5060921376.528326',
 'last but not least, the model score is0.6140090808478247')
```


```
model = LinearRegression()
check_model (model, X_train, X_test, y_train, y_test)
```

```
('mean absolute error is:50888.819009882776',
 'r2 score is:0.4106975255730O443',
 'root mse is:5060921376.528326',
 'last but not least, the model score is0.6140090808478247')
```

Scaling du model en Distribution normale

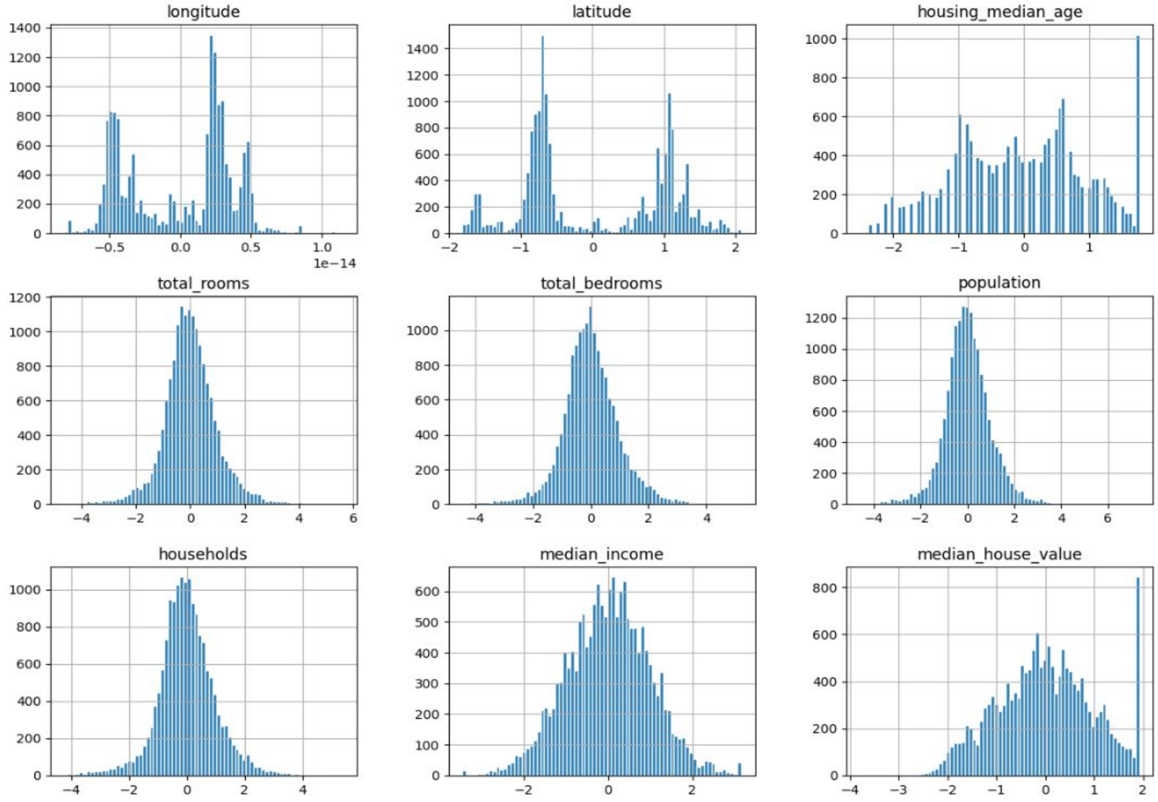
```
from sklearn.preprocessing import PowerTransformer
pt = PowerTransformer()
new_data = pt.fit_transform(data)
```

```
new_data = pd.DataFrame(new_data)
new_data
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	blue_near
0	-7.494005e-16	0.689303	-1.926087	-0.223616	0.074415	0.223832	-0.015268	-1.773871	-1.533553	0.431417
1	3.302913e-15	-0.939986	-1.724793	-0.097700	-0.355465	-0.452196	-0.303370	1.502768	0.757965	-1.046702
2	-1.387779e-15	0.620841	-0.241438	-1.160705	-0.990031	-0.349338	-0.907024	-1.683667	-1.861716	0.431417
3	2.220446e-15	-0.661816	0.228659	-1.539490	-0.906125	-0.918574	-0.833792	-1.714932	0.182829	-1.046702
4	-3.247402e-15	1.061092	-0.566528	-0.173991	-0.263837	-0.082398	-0.178151	0.340322	-0.732273	0.431417
...	...	...	...	...	...	...	...	...	...	...
16507	-4.357625e-15	1.582017	-0.649535	-0.533411	-0.658994	-0.642124	-0.711694	-0.287555	-1.109935	0.431417
16508	-4.940492e-15	1.165243	1.549131	0.115013	0.167811	-0.081152	0.121853	-0.627716	-0.955111	1.594333
16509	-3.219647e-15	1.405650	-0.733315	0.212355	0.036972	0.009996	0.025891	0.553618	0.115531	0.431417
16510	2.581269e-15	-0.623808	0.829704	0.347236	0.967984	1.175605	0.945858	-0.659050	-0.282906	-1.046702
16511	-7.827072e-15	1.856775	0.228659	-0.018088	0.040121	0.071807	-0.004895	-0.944334	-1.344664	1.160847

16512 rows × 10 columns

Analyse de la distribution des features



Nouveau modèle de référence

```
X = new_data[['longitude','latitude','housing_median_age','total_rooms','median_income','blue_near','population']]
y = new_data['median_house_value']
```

```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3, random_state = 0)
```

```
model = LinearRegression()
check_model (model, X_train, X_test, y_train, y_test)
```

```
('mean absolute error is:0.45594435650877463',
 'r2 score is:0.4414370696328561',
 'root mse is:0.36320764489167673',
 'last but not least, the model score is0.6295365155814411')
```

```
# model
model = LinearRegression()
# 5 - Fold Cross validate model
cv_results = cross_validate(model, X, y, cv =5)

cv_results['test_score'].mean()
```

```
0.6475397912446548
```