Université Abdelmalek Essaadi
Faculté ses Sciences et techniques de Tanger
Département Génie Informatique

Réalisé par:

- **Fatima Zahrae Khayat**
- **Ikram Ait-Kaddi**

Encadré par:

Pr. EL AACHAk LOTFI

Remerciement :

Au terme de ce travail, nous tenons à exprimer notre profonde gratitude à notre cher professeur **Pr. EL AACHAk LOTFI** pour votre suivi et pour votre énorme soutien, qui n'a cessé de nous prodiguer et de nous guidé tout au long de ce semestre, ainsi pour votre générosité en matière de formation et d'encadrement.

Nous tenons à vous remercier monsieur de nous avoir incités à travailler en mettant à notre disposition votre expériences et votre compétence.

But de projet :

Demandez à un acheteur de maison de décrire la maison de ses rêves, et il ne commencera probablement pas par la hauteur du plafond du sous-sol ou la proximité d'une voie ferrée est-ouest. Mais l'ensemble de données de ce concours de terrain de jeu prouve que beaucoup plus influence les négociations de prix que le nombre de chambres ou une clôture blanche.

Avec 79 variables explicatives décrivant (presque) tous les aspects des maisons résidentielles à Ames, Iowa, ce projet est sert à mettre au défi de prédire le prix final de chaque maison.

PLAN

5- Préparation des données pour la modélisation

6 - Modélisation

- Random forest Regressor
- Régression linéaire
- Lasso regression
- XGB Regressor
- SVR : support de régression vectorielle

7- Choisir le meilleur modèle

8- vérification de l'existence d'un sur ajustement ou d'un sous-ajustement

9- Submission

1- Introduction

L'apprentissage automatique est un sous-domaine de l'intelligence artificielle (IA). Leur 'objectif est de comprendre la structure des données et d'intégrer ces données dans des modèles qui peuvent être compris et utilisés par les gens.

Bien que l'apprentissage automatique soit un domaine de l'informatique, il diffère des approches informatiques traditionnelles. Dans l'informatique traditionnelle, les algorithmes sont des ensembles d'instructions explicitement programmées utilisées par les ordinateurs pour calculer ou résoudre des problèmes. Ils permettent à la place aux ordinateurs de s'entraîner sur les entrées de données et d'utiliser l'analyse statistique afin de générer des valeurs comprises dans une plage spécifique. Pour cette raison, l'apprentissage automatique facilite la création de modèles par les ordinateurs à partir d'échantillons de données afin d'automatiser les processus de prise de décision basés sur les entrées de données.

Aujourd'hui, tout utilisateur de technologie a bénéficié de l'apprentissage automatique. La technologie de reconnaissance faciale permet aux plateformes de médias sociaux d'aider les utilisateurs à marquer et à partager des photos d'amis. Les moteurs de recommandation, alimentés par l'apprentissage automatique, suggèrent les films ou les émissions de télévision à regarder ensuite en fonction des préférences de l'utilisateur. Les voitures autonomes qui s'appuient sur l'apprentissage automatique pour naviguer pourraient bientôt être disponibles pour les consommateurs…….

Dans ce didacticiel, nous examinerons les méthodes d'apprentissage automatique courantes de l'apprentissage supervisé y compris la régression, El quelque modèle tel que Random forestRegressor, regression linear, Lasso Regression….

2- Exploration des données

❖ Importation des bibliothèques nécessaires:

```
# Load librarie
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.feature_selection import VarianceThreshold
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import RFE
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
```

```
df_test = pd.read_csv("C:/Users/DELL/Desktop/machine_learnig/dataset/house/test.csv")
df_train = pd.read_csv("C:/Users/DELL/Desktop/machine_learnig/dataset/house/train.csv")
df_train.head()
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal	MoS
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0	

5 rows × 81 columns

❖ Informations sur chaque colonne dans df_train

```
# info of each of the variables in our train set
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     1452 non-null   object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
 28  ExterCond      1460 non-null   object
 29  Foundation     1460 non-null   object
 30  BsmtQual       1423 non-null   object
 31  BsmtCond       1423 non-null   object
```

```
 32  BsmtExposure   1422 non-null   object
 33  BsmtFinType1   1423 non-null   object
 34  BsmtFinSF1     1460 non-null   int64
 35  BsmtFinType2   1422 non-null   object
 36  BsmtFinSF2     1460 non-null   int64
 37  BsmtUnfSF      1460 non-null   int64
 38  TotalBsmtSF    1460 non-null   int64
 39  Heating        1460 non-null   object
 40  HeatingQC      1460 non-null   object
 41  CentralAir     1460 non-null   object
 42  Electrical     1459 non-null   object
 43  1stFlrSF       1460 non-null   int64
 44  2ndFlrSF       1460 non-null   int64
 45  LowQualFinSF   1460 non-null   int64
 46  GrLivArea      1460 non-null   int64
 47  BsmtFullBath   1460 non-null   int64
 48  BsmtHalfBath   1460 non-null   int64
 49  FullBath       1460 non-null   int64
 50  HalfBath       1460 non-null   int64
 51  BedroomAbvGr   1460 non-null   int64
 52  KitchenAbvGr   1460 non-null   int64
 53  KitchenQual    1460 non-null   object
 54  TotRmsAbvGrd   1460 non-null   int64
 55  Functional     1460 non-null   object
 56  Fireplaces     1460 non-null   int64
 57  FireplaceQu    770 non-null    object
 58  GarageType     1379 non-null   object
 59  GarageYrBlt    1379 non-null   float64
 60  GarageFinish   1379 non-null   object
 61  GarageCars     1460 non-null   int64
 62  GarageArea     1460 non-null   int64
 63  GarageQual     1379 non-null   object
 64  GarageCond     1379 non-null   object
 65  PavedDrive     1460 non-null   object
 66  WoodDeckSF     1460 non-null   int64
 67  OpenPorchSF    1460 non-null   int64
 68  EnclosedPorch  1460 non-null   int64
 69  3SsnPorch      1460 non-null   int64
 70  ScreenPorch    1460 non-null   int64
 71  PoolArea       1460 non-null   int64
 72  PoolQC         7 non-null      object
 73  Fence          281 non-null    object
 74  MiscFeature    54 non-null     object
 75  MiscVal        1460 non-null   int64
 76  MoSold         1460 non-null   int64
 77  YrSold         1460 non-null   int64
 78  SaleType       1460 non-null   object
 79  SaleCondition  1460 non-null   object
 80  SalePrice      1460 non-null   int64
dtypes: float64(3), int64(35), object(43)
```

```python
# Drop the 'Id' column from the train set
df_train.drop(["Id"], axis=1, inplace=True)

# Save the list of 'Id' before dropping it from the test set
Id_test_list = df_test["Id"].tolist()
df_test.drop(["Id"], axis=1, inplace=True)
```

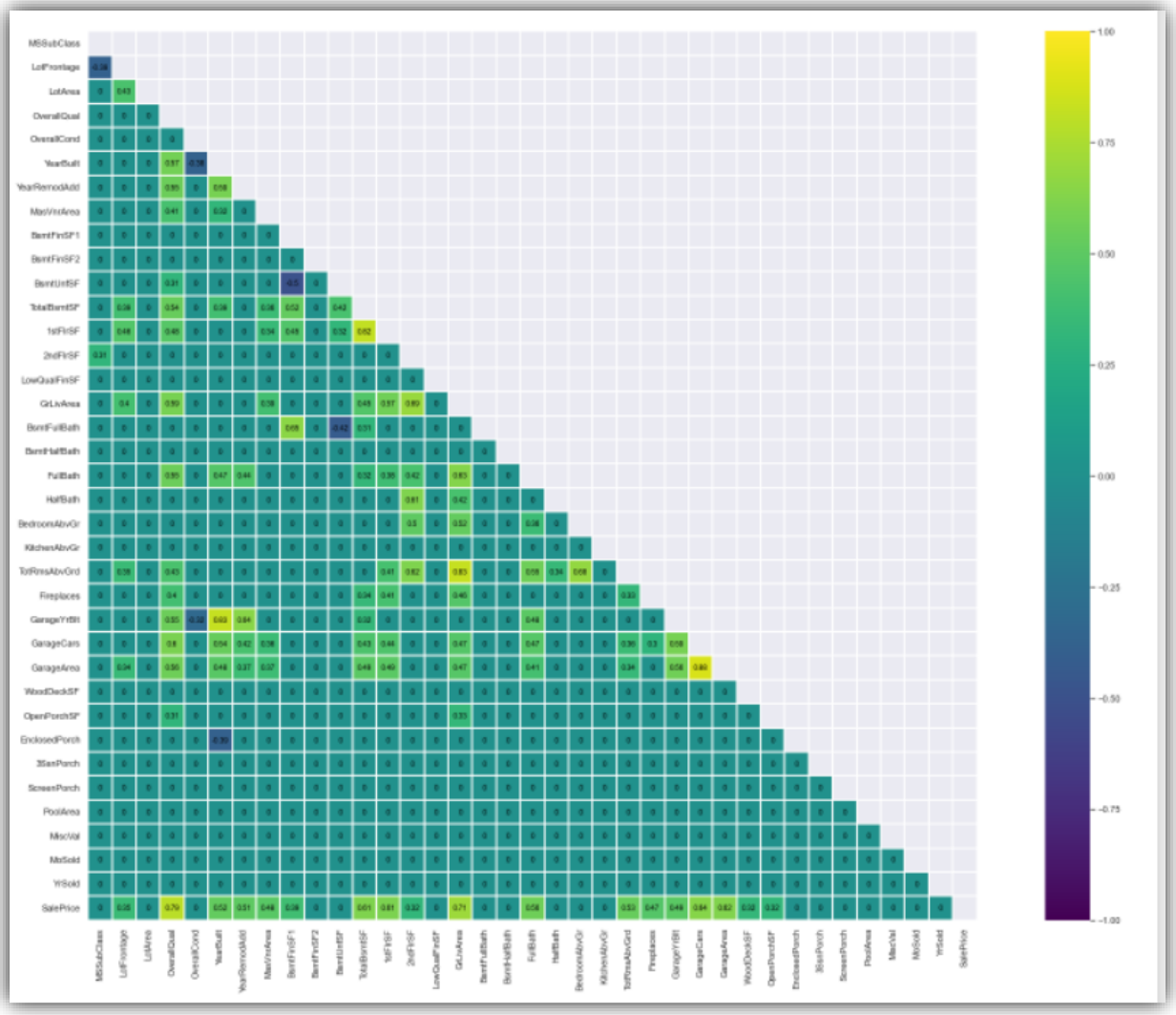
3- Analyse et nettoyage des données

3-1- Numerical features

```
# Let's select the columns of the train set with numerical data
df_train_num = df_train.select_dtypes(exclude=["object"])
```

- La matrice de corrélation ne considère pas les valeurs nulles

```
# Heatmap for all the remaining numerical data including the taget 'SalePrice'

# Define the heatmap parameters
pd.options.display.float_format = "{:,.2f}".format

# Define correlation matrix
corr_matrix = df_train_num.corr()

# Replace correlation < |0.3| by 0 for a better visibility
corr_matrix[(corr_matrix < 0.3) & (corr_matrix > -0.3)] = 0

# Mask the upper part of the heatmap
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Choose the color map
cmap = "viridis"
plt.figure(figsize = (30,20))
# plot the heatmap
sns.heatmap(corr_matrix, mask=mask, vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot_kws={"size": 9, "color": "black"}, square=True, cmap=cmap, annot=True)
```



❖ Caractéristique qui ont une corrélation entre eux supérieure à 0,7

```
corr_matrix = df_train_num.corr().abs()

#the matrix is symmetric so we need to extract upper triangle matrix without diagonal (k = 1)

sol = (corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
                  .stack()
                  .sort_values(ascending=False))
sol = sol.to_frame()
sol.columns=['corr']
sol[sol['corr'] >  0.7]
```

```
C:\Users\Dell\AppData\Local\Temp/ipykernel_1820/1198977606.py:5: DeprecationWarning: `np.bool` is a deprecated alias for the bu
iltin `bool`. To silence this warning, use `bool` by itself. Doing this will not modify any behavior and is safe. If you specif
ically wanted the numpy scalar type, use `np.bool_` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  sol = (corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
```

		corr
GarageCars	GarageArea	0.88
YearBuilt	GarageYrBlt	0.83
GrLivArea	TotRmsAbvGrd	0.83
TotalBsmtSF	1stFlrSF	0.82
OverallQual	SalePrice	0.79
GrLivArea	SalePrice	0.71

Nous prendrons cela en considération lors de la suppression des fonctionnalités, car si nous avons une corrélation entre les fonctionnalités de remorquage, nous devons supprimer l'une d'entre elles, nous avons donc besoin de plus d'informations pour savoir quelle est la fonctionnalité à supprimer

3-1-2 Relation avec salePrice dans train dataset

Remarque:

Les caractéristiques qui sont corrélées avec salePrice, elles nous aideront à faire la prédiction

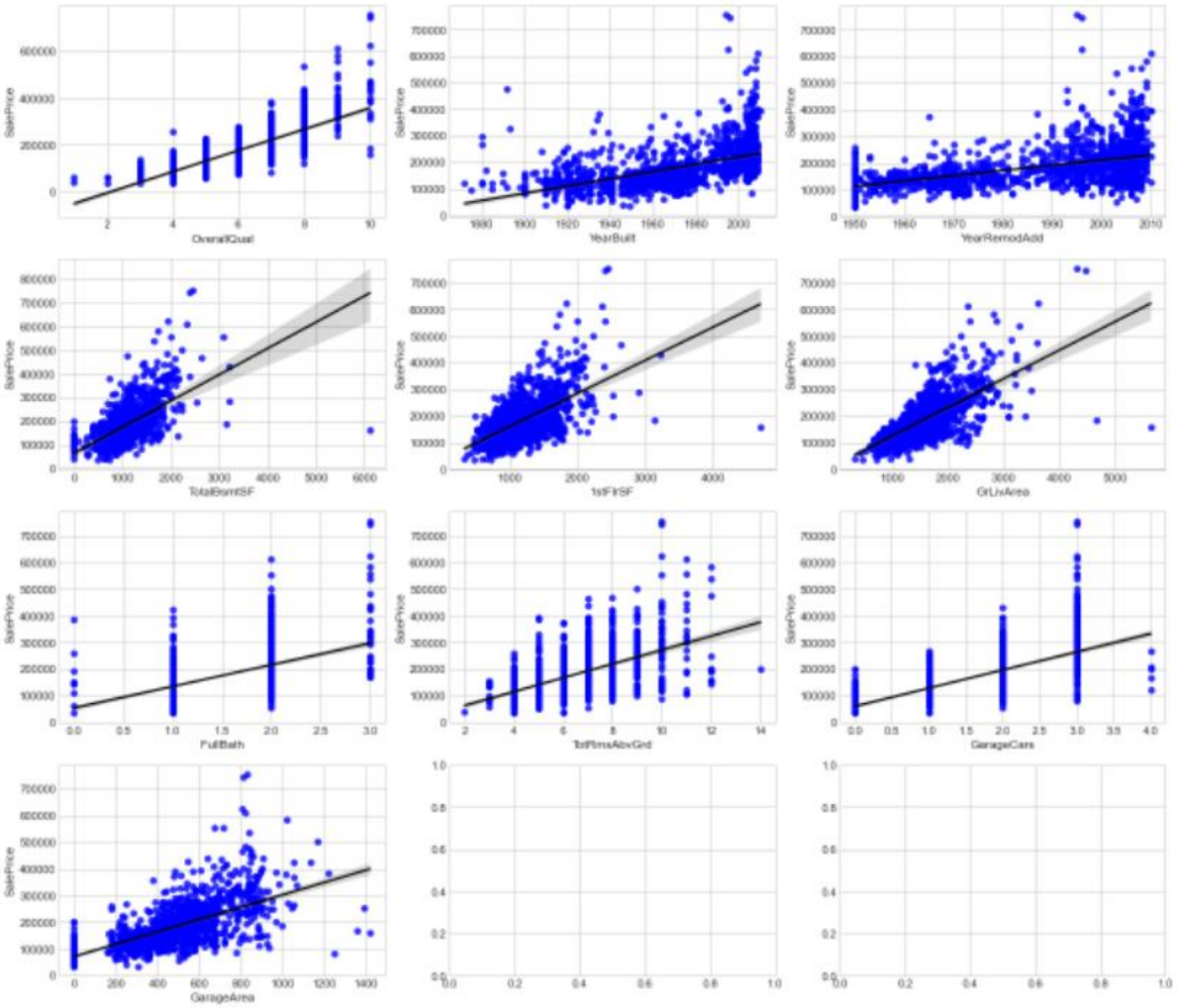
```
# Let's select features where the correlation with 'SalePrice' is higher than |0.3|
# -1 because the latest row is SalePrice
df_num_corr = df_train_num.corr()["SalePrice"][:-1]

# Correlated features (r2 > 0.5)
high_features_list = df_num_corr[abs(
    df_num_corr) >= 0.5].sort_values(ascending=False)
print(
    f"{len(high_features_list)} strongly correlated values with SalePrice:\n{high_features_list}\n")

# Correlated features (0 < r2 < 0.3)
low_features_list = df_num_corr[(abs(df_num_corr) < 0.5) & (abs(df_num_corr) >= 0.3)].sort_values(ascending=False)
print(
    f"{len(low_features_list)} slightly correlated values with SalePrice:\n{low_features_list}")
```

```
10 strongly correlated values with SalePrice:
OverallQual    0.79
GrLivArea      0.71
GarageCars     0.64
GarageArea     0.62
TotalBsmtSF    0.61
1stFlrSF       0.61
FullBath       0.56
TotRmsAbvGrd   0.53
YearBuilt      0.52
YearRemodAdd   0.51
Name: SalePrice, dtype: float64

8 slightly correlated values with SalePrice:
GarageYrBlt   0.49
MasVnrArea    0.48
Fireplaces    0.47
BsmtFinSF1    0.39
LotFrontage   0.35
WoodDeckSF    0.32
2ndFlrSF      0.32
OpenPorchSF   0.32
Name: SalePrice, dtype: float64
```

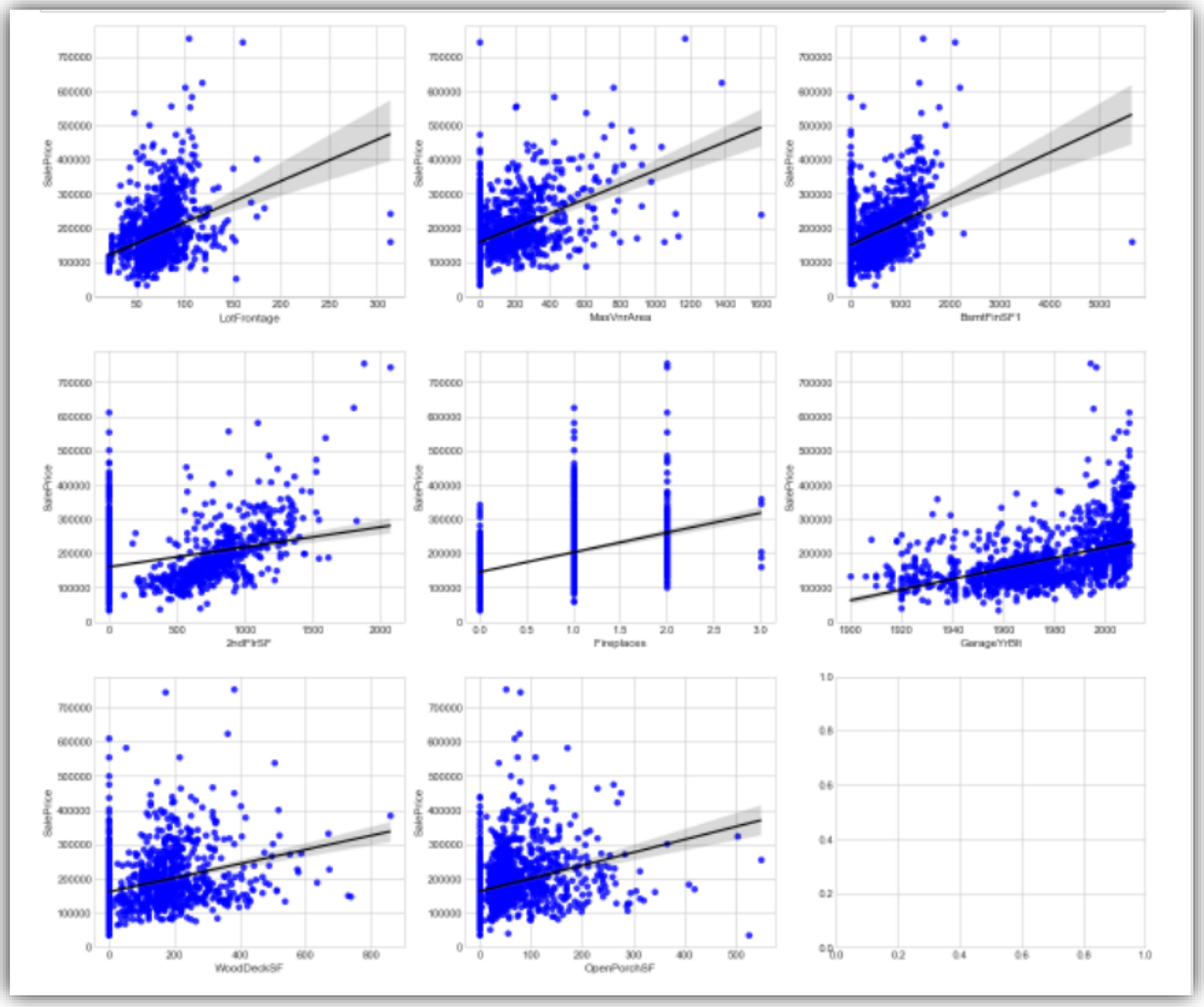
Remarque:

Nous laisserons comme valeurs numériques dans notre jeu de données celles qui ont la relation avec le prix de vente

```
# Features with high correlation (higher than 0.5)
strong_features = df_num_corr[abs(df_num_corr) >= 0.5].index.tolist()
strong_features.append("SalePrice")

df_strong_features = df_train_num.loc[:, strong_features]

plt.style.use("seaborn-whitegrid")  # define figures style
fig, ax = plt.subplots(round(len(strong_features) / 3), 3)

for i, ax in enumerate(fig.axes):
    # plot the correlation of each feature with SalePrice
    if i < len(strong_features)-1:
        sns.regplot(x=strong_features[i], y="SalePrice", data=df_strong_features, ax=ax, scatter_kws={
                    "color": "blue"}, line_kws={"color": "black"})
```



```
# Features with low correlation (between 0.3 and 0.5)
low_features = df_num_corr[(abs(df_num_corr) >= 0.3) & (
    abs(df_num_corr) < 0.5)].index.tolist()
low_features.append("SalePrice")

df_low_features = df_train_num.loc[:, low_features]

plt.style.use("seaborn-whitegrid")  # define figures style
fig, ax = plt.subplots(round(len(low_features) / 3), 3)

for i, ax in enumerate(fig.axes):
    # plot the correlation of each feature with SalePrice
    if i < len(low_features) - 1:
        sns.regplot(x=low_features[i], y="SalePrice", data=df_low_features, ax=ax, scatter_kws={
                    "color": "blue"}, line_kws={"color": "black"},)
```



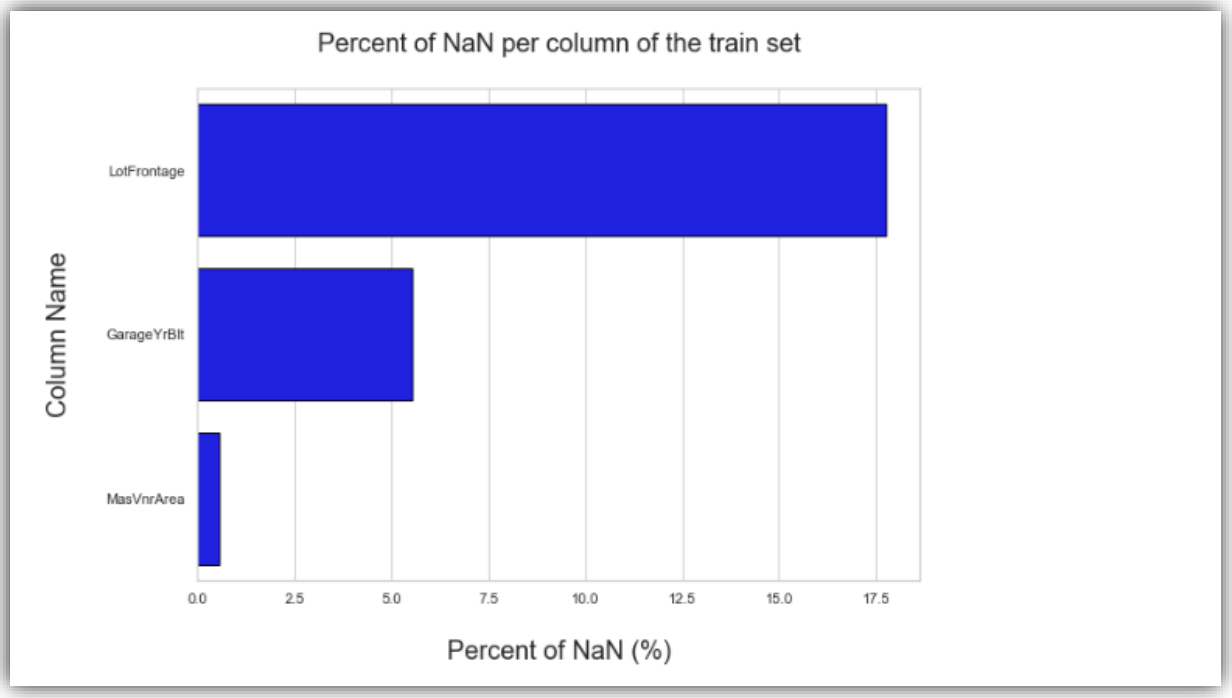
```
# Define the list of numerical fetaures to keep
list_of_numerical_features = strong_features[:-1] + low_features

# Let's select these features form our train set
df_train_num = df_train_num.loc[:, list_of_numerical_features]

# The same features are selected from the test set (-1 -> except 'SalePrice')
df_test_num = df_test.loc[:, list_of_numerical_features[:-1]]
df_train_num.head()
```

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea	LotFrontage	MasVnrArea	Bsn
0	7	2003	2003	856	856	1710	2	8	2	548	65.00	196.00	
1	6	1976	1976	1262	1262	1262	2	6	2	460	80.00	0.00	
2	7	2001	2002	920	920	1786	2	6	2	608	68.00	162.00	
3	7	1915	1970	756	961	1717	1	7	3	642	60.00	0.00	
4	8	2000	2000	1145	1145	2198	2	9	3	836	84.00	350.00	

3-1-3 Missing values dans train dataset

```
# Check the NaN of the train set by ploting percent of missing values per column
column_with_nan = df_train_num.columns[df_train_num.isnull().any()]
column_name = []
percent_nan = []

for i in column_with_nan:
    column_name.append(i)
    percent_nan.append(
        round(df_train_num[i].isnull().sum()*100/len(df_train_num), 2))

tab = pd.DataFrame(column_name, columns=["Column"])
tab["Percent_NaN"] = percent_nan
tab.sort_values(by=["Percent_NaN"], ascending=False, inplace=True)


# Define figure parameters
sns.set(rc={"figure.figsize": (10, 7)})
sns.set_style("whitegrid")

# Plot results
p = sns.barplot(x="Percent_NaN", y="Column", data=tab,
                edgecolor="black", color="blue")

p.set_title("Percent of NaN per column of the train set\n", fontsize=20)
p.set_xlabel("\nPercent of NaN (%)", fontsize=20)
p.set_ylabel("Column Name\n", fontsize=20)
```

```
Text(0, 0.5, 'Column Name\n')
```

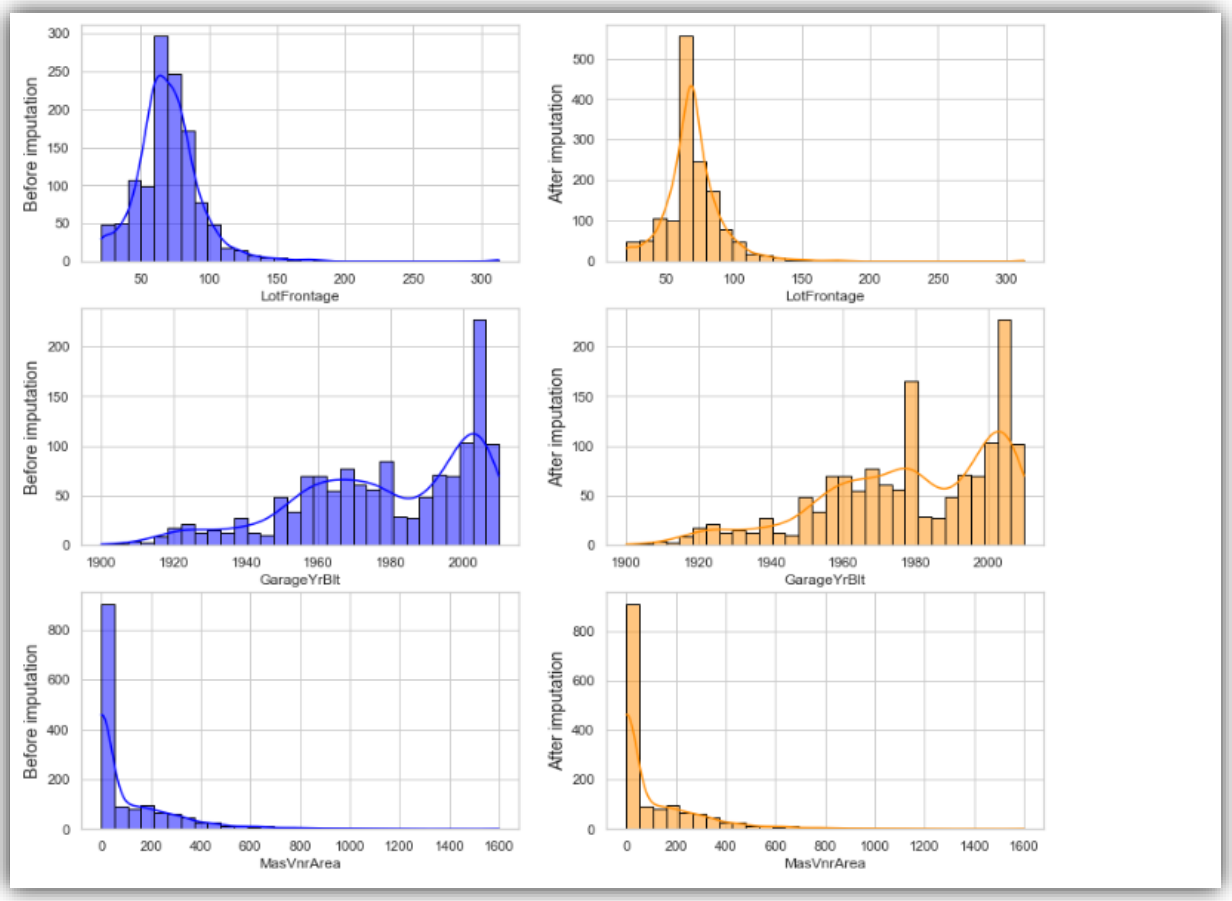


- Imputation des valeurs manquantes

```
# Imputation of missing values (NaNs) with SimpleImputer
my_imputer = SimpleImputer(strategy="median")
df_train_imputed = pd.DataFrame(my_imputer.fit_transform(df_train_num))
df_train_imputed.columns = df_train_num.columns
```

3-1-4 vérification de la distribution de chaque caractéristique imputée avant et après l'imputation.

```python
# Define figure parameters
sns.set(rc={"figure.figsize": (14, 12)})
sns.set_style("whitegrid")
fig, axes = plt.subplots(3, 2)

# Plot the results
for feature, fig_pos in zip(["LotFrontage", "GarageYrBlt", "MasVnrArea"], [0, 1, 2]):

    """Features distribution before and after imputation"""

    # before imputation
    p = sns.histplot(ax=axes[fig_pos, 0], x=df_train_num[feature],
                     kde=True, bins=30, color="blue", edgecolor="black")
    p.set_ylabel(f"Before imputation", fontsize=14)

    # after imputation
    q = sns.histplot(ax=axes[fig_pos, 1], x=df_train_imputed[feature],
                     kde=True, bins=30, color="darkorange", edgecolor="black")
    q.set_ylabel(f"After imputation", fontsize=14)
```



Pour "LotFrontage" et "GarageYrBlt", les distributions ont changé après imputations. Il y a une sur-représentation de la classe médiane par rapport à la distribution originale. Cependant, la distribution reste la même pour "MasVnrArea". Ainsi, pour éviter toute erreur liée à l'imputation je ne garde que la caractéristique "MasVnrArea" pour mes analyses.

```
# Drop 'LotFrontage' and 'GarageYrBlt'
df_train_imputed.drop(["LotFrontage", "GarageYrBlt"], axis=1, inplace=True)
df_train_imputed.head()
```

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea	MasVnrArea	BsmtFinSF1	2nd
0	7.00	2,003.00	2,003.00	856.00	856.00	1,710.00	2.00	8.00	2.00	548.00	196.00	706.00	8
1	6.00	1,976.00	1,976.00	1,262.00	1,262.00	1,262.00	2.00	6.00	2.00	460.00	0.00	978.00	
2	7.00	2,001.00	2,002.00	920.00	920.00	1,786.00	2.00	6.00	2.00	608.00	162.00	486.00	8
3	7.00	1,915.00	1,970.00	756.00	961.00	1,717.00	1.00	7.00	3.00	642.00	0.00	216.00	7
4	8.00	2,000.00	2,000.00	1,145.00	1,145.00	2,198.00	2.00	9.00	3.00	836.00	350.00	655.00	1,0

3-1-5 feature importance

```
column_name = []


for i in df_train_imputed:
    column_name.append(i)
print(column_name)

array = df_train_imputed.values
X = array[:,0:15]
Y = array[:,16]

# feature extraction
model = ExtraTreesClassifier()
model.fit(X, Y)
print(model.feature_importances_)
```

```
['OverallQual', 'YearBuilt', 'YearRemodAdd', 'TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'FullBath', 'TotRmsAbvGrd', 'GarageCars',
'GarageArea', 'MasVnrArea', 'BsmtFinSF1', '2ndFlrSF', 'Fireplaces', 'WoodDeckSF', 'OpenPorchSF', 'SalePrice']
[0.05545658 0.08868852 0.08371813 0.09225945 0.09207915 0.09396016
 0.0201888  0.06267328 0.02767086 0.09039202 0.05746339 0.08457288
 0.05056497 0.03399914 0.06631266]
```

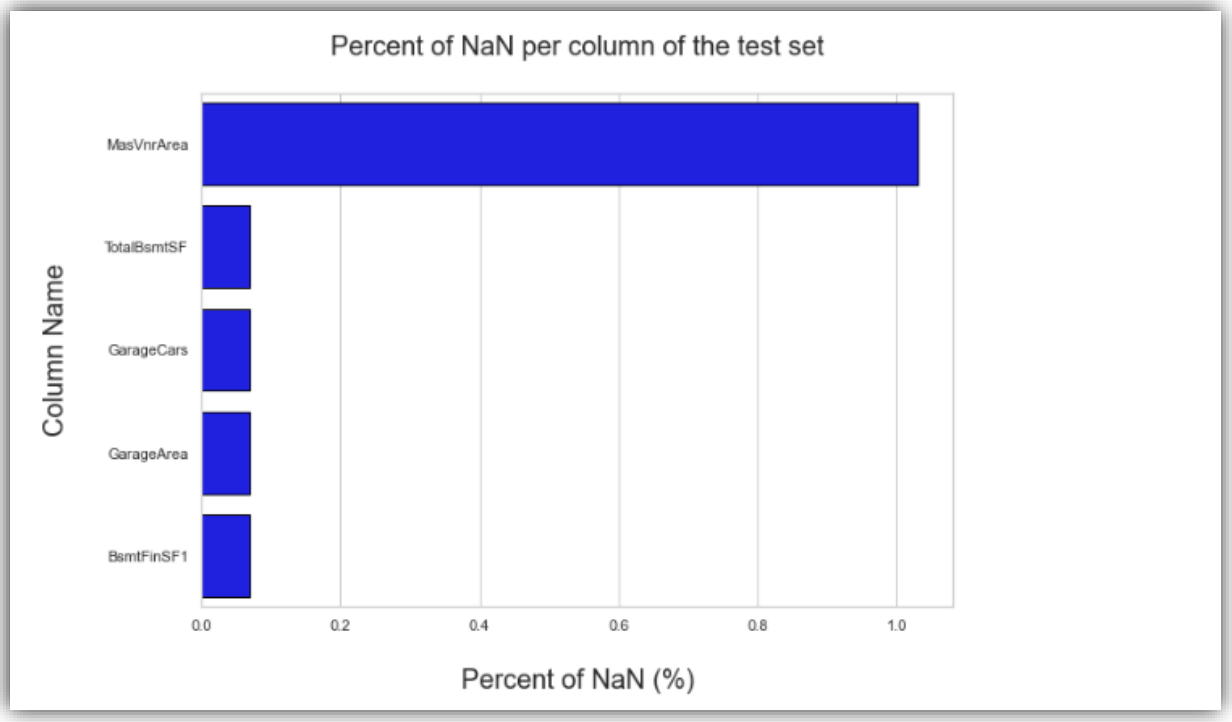
En utilisant la méthode d'importance des caractéristiques et trouve que GrLivArea est l'attribut le plus important, après avoir 1stFlrSF et totalBsmtSF et GrageArea, nous avons déjà montré que ces attributs ont une forte corrélation avec l'attribut salePrice.

3-1-6 Missing value dans Test dataset

Les colonnes qui ont été supprimées dans la rame doivent également être supprimées dans l'ensemble de test afin que les deux ensembles de données restent identiques pour la modélisation et la prédiction.

```
# Drop the same features from test set as for the train set
df_test_num.drop(["LotFrontage", "GarageYrBlt"], axis=1, inplace=True)
```

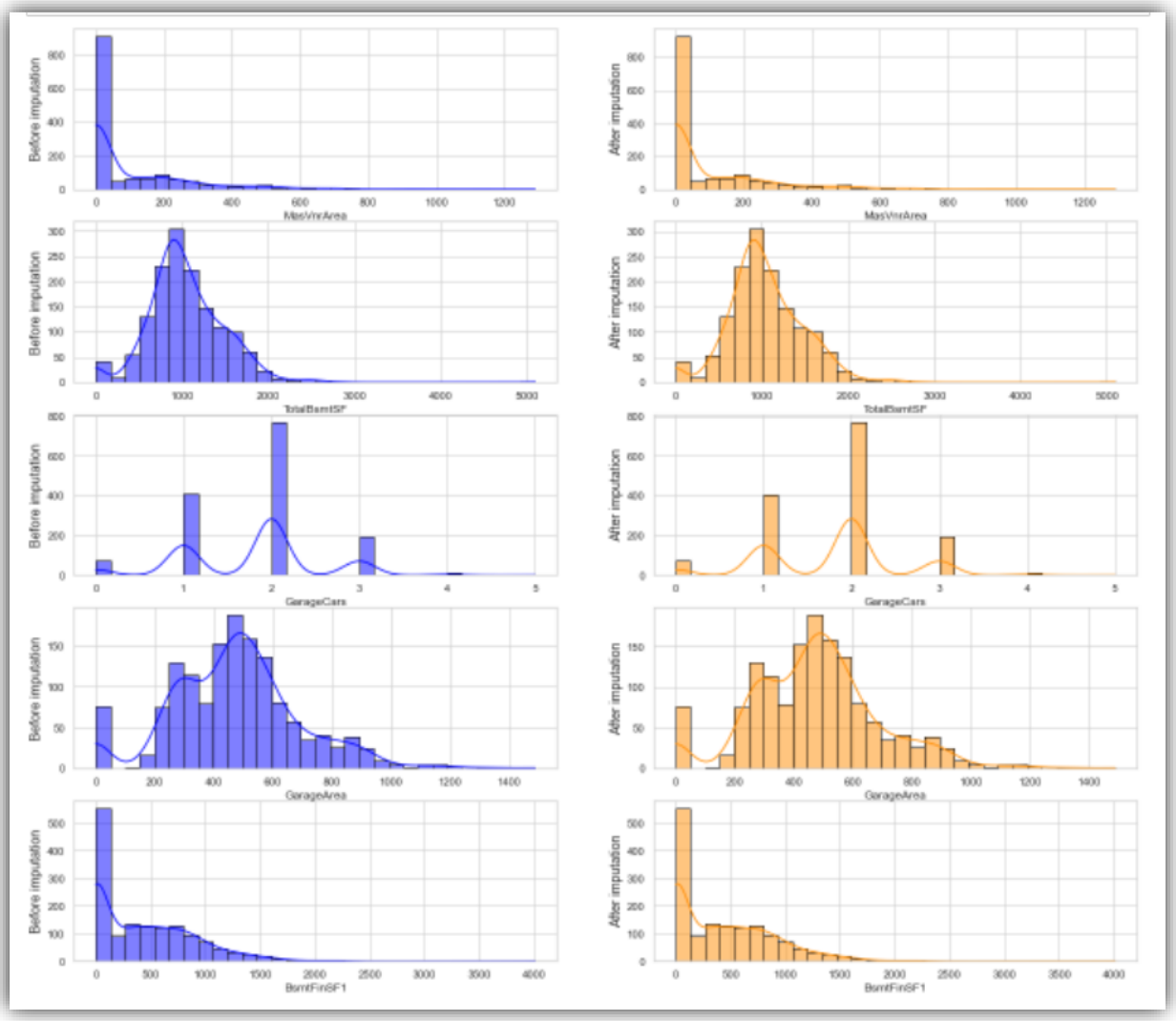
```
# Check the NaN of the test set by ploting percent of missing values per column
column_with_nan = df_test_num.columns[df_test_num.isnull().any()]
column_name = []
percent_nan = []

for i in column_with_nan:
    column_name.append(i)
    percent_nan.append(
        round(df_test_num[i].isnull().sum()*100/len(df_test_num), 2))

tab = pd.DataFrame(column_name, columns=["Column"])
tab["Percent_NaN"] = percent_nan
tab.sort_values(by=["Percent_NaN"], ascending=False, inplace=True)


# Define figure parameters
sns.set(rc={"figure.figsize": (10, 7)})
sns.set_style("whitegrid")

# Plot results
p = sns.barplot(x="Percent_NaN", y="Column", data=tab,
                edgecolor="black", color="blue")

p.set_title("Percent of NaN per column of the test set\n", fontsize=20)
p.set_xlabel("\nPercent of NaN (%)", fontsize=20)
p.set_ylabel("Column Name\n", fontsize=20)
```



- Imputation dans test dataset

```
# Imputation of missing values (NaNs) with SimpleImputer
my_imputer = SimpleImputer(strategy="median")
df_test_imputed = pd.DataFrame(my_imputer.fit_transform(df_test_num))
df_test_imputed.columns = df_test_num.columns
```

```
# Let's check the distribution of each imputed feature before and after imputation

# Define figure parameters
sns.set(rc={"figure.figsize": (20, 18)})
sns.set_style("whitegrid")
fig, axes = plt.subplots(5, 2)

# Plot the results
for feature, fig_pos in zip(tab["Column"].tolist(), range(0, 6)):

    """Features distribution before and after imputation"""

    # before imputation
    p = sns.histplot(ax=axes[fig_pos, 0], x=df_test_num[feature],
                     kde=True, bins=30, color="blue", edgecolor="black")
    p.set_ylabel(f"Before imputation", fontsize=14)

    # after imputation
    q = sns.histplot(ax=axes[fig_pos, 1], x=df_test_imputed[feature],
                     kde=True, bins=30, color="darkorange", edgecolor="black",)
    q.set_ylabel(f"After imputation", fontsize=14)
```



Le pourcentage de NaN dans chacune de ces caractéristiques ne dépasse pas 1,5 %. Ainsi, en imputant ces données manquantes, peu d'erreurs ont été introduites et les distributions sont similaires avant et après imputation.

6-3 Normalisation des attributs

```
scaler = StandardScaler()

#train
print(scaler.fit(df_train_imputed.iloc[:,0:16]))
print(scaler.mean_)
df_train_imputed.iloc[:,0:16]=scaler.transform(df_train_imputed.iloc[:,0:16])
# test
scaler.fit(df_test_imputed)
print(scaler.mean_)
df_test_imputed=scaler.transform(df_test_imputed)
```

```
StandardScaler()
[6.09931507e+00 1.97126781e+03 1.98486575e+03 1.05742945e+03
 1.16262671e+03 1.51546370e+03 1.56506849e+00 6.51780822e+00
 1.76712329e+00 4.72980137e+02 1.03117123e+02 4.43639726e+02
 3.46992466e+02 6.13013699e-01 9.42445205e+01 4.66602740e+01]
[6.07882111e+00 1.97135778e+03 1.98366278e+03 1.04607814e+03
 1.15653461e+03 1.48604592e+03 1.57093900e+00 6.38519534e+00
 1.76627827e+00 4.72773818e+02 9.96737491e+01 4.39142906e+02
 3.25967786e+02 5.81220014e-01 9.31747772e+01 4.83139136e+01]
```

```
df_train_imputed.head()
```

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea	MasVnrArea	BsmtFinSF1	2nd
0	0.65	1.05	0.88	-0.46	-0.79	0.37	0.79	0.91	0.31	0.35	0.51	0.58	
1	-0.07	0.16	-0.43	0.47	0.26	-0.48	0.79	-0.32	0.31	-0.06	-0.57	1.17	
2	0.65	0.98	0.83	-0.31	-0.63	0.52	0.79	-0.32	0.31	0.63	0.33	0.09	
3	0.65	-1.86	-0.72	-0.69	-0.52	0.38	-1.03	0.30	1.65	0.79	-0.57	-0.50	
4	1.37	0.95	0.73	0.20	-0.05	1.30	0.79	1.53	1.65	1.70	1.37	0.46	

```
names = ['OverallQual', 'YearBuilt', 'YearRemodAdd', 'TotalBsmtSF', '1stFlrSF', 'GrLivArea', 'FullBath', 'TotRmsAbvGrd', 'Garage
```

```
pd.DataFrame(df_test_imputed, columns = names).head()
```

	OverallQual	YearBuilt	YearRemodAdd	TotalBsmtSF	1stFlrSF	GrLivArea	FullBath	TotRmsAbvGrd	GarageCars	GarageArea	MasVnrArea	BsmtFinSF1	2nd
0	-0.75	-0.34	-1.07	-0.37	-0.65	-1.22	-1.03	-0.92	-0.99	1.19	-0.56	0.06	
1	-0.05	-0.44	-1.21	0.64	0.43	-0.32	-1.03	-0.26	-0.99	-0.74	0.05	1.06	
2	-0.75	0.84	0.68	-0.27	-0.57	0.29	0.77	-0.26	0.30	0.04	-0.56	0.77	
3	-0.05	0.88	0.68	-0.27	-0.58	0.24	0.77	0.41	0.30	-0.01	-0.45	0.36	
4	1.34	0.68	0.39	0.53	0.31	-0.42	0.77	-0.92	0.30	0.15	-0.56	-0.39	

3-2- Categorical features

3-2-1 Exploration et nettoyage des Categorical features

```
cat =[
    i for i in df_train.columns if df_train.dtypes[i] == "object"]
# test dataset
df_test_cat = df_test[cat]
# train dataset

df_train_cat = df_train[cat]
df_train_cat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 43 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   MSZoning        1460 non-null   object
 1   Street          1460 non-null   object
 2   Alley           91 non-null     object
 3   LotShape        1460 non-null   object
 4   LandContour     1460 non-null   object
 5   Utilities       1460 non-null   object
 6   LotConfig       1460 non-null   object
 7   LandSlope       1460 non-null   object
 8   Neighborhood    1460 non-null   object
 9   Condition1      1460 non-null   object
 10  Condition2      1460 non-null   object
 11  BldgType        1460 non-null   object
 12  HouseStyle      1460 non-null   object
 13  RoofStyle       1460 non-null   object
 14  RoofMatl        1460 non-null   object
 15  Exterior1st     1460 non-null   object
 16  Exterior2nd     1460 non-null   object
 17  MasVnrType      1452 non-null   object
 18  ExterQual       1460 non-null   object
 19  ExterCond       1460 non-null   object
 20  Foundation      1460 non-null   object
 21  BsmtQual        1423 non-null   object
 22  BsmtCond        1423 non-null   object
 23  BsmtExposure    1422 non-null   object
 24  BsmtFinType1    1423 non-null   object
 25  BsmtFinType2    1422 non-null   object
 26  Heating         1460 non-null   object
 27  HeatingQC       1460 non-null   object
 28  CentralAir      1460 non-null   object
 29  Electrical      1459 non-null   object
 30  KitchenQual     1460 non-null   object
 31  Functional      1460 non-null   object
 32  FireplaceQu     770 non-null    object
 33  GarageType      1379 non-null   object
 34  GarageFinish    1379 non-null   object
 35  GarageQual      1379 non-null   object
 36  GarageCond      1379 non-null   object
 37  PavedDrive      1460 non-null   object
 38  PoolQC          7 non-null      object
 39  Fence           281 non-null    object
 40  MiscFeature     54 non-null     object
 41  SaleType        1460 non-null   object
 42  SaleCondition   1460 non-null   object
dtypes: object(43)
memory usage: 490.6+ KB
```

❖ Affichage des valeurs des features :

```
s = []
i = 0
for k in df_train_cat.select_dtypes(include=["object"]):
    print(k, df_train_cat[k].unique())

    s.append(df_train_cat[k].unique())
    for j in s:
        for m in j:
           # print(m)

            s=[]
#d = {1: 'red', 2: 'orange', 3: 'yellow'}
#lst = [d[k] for k in lst]
```

```
MSZoning ['RL' 'RM' 'C (all)' 'FV' 'RH']
Street ['Pave' 'Grvl']
Alley [nan 'Grvl' 'Pave']
LotShape ['Reg' 'IR1' 'IR2' 'IR3']
LandContour ['Lvl' 'Bnk' 'Low' 'HLS']
Utilities ['AllPub' 'NoSeWa']
LotConfig ['Inside' 'FR2' 'Corner' 'CulDSac' 'FR3']
LandSlope ['Gtl' 'Mod' 'Sev']
Neighborhood ['CollgCr' 'Veenker' 'Crawfor' 'NoRidge' 'Mitchel' 'Somerst' 'NWAmes'
 'OldTown' 'BrkSide' 'Sawyer' 'NridgHt' 'NAmes' 'SawyerW' 'IDOTRR'
 'MeadowV' 'Edwards' 'Timber' 'Gilbert' 'StoneBr' 'ClearCr' 'NPkVill'
 'Blmngtn' 'BrDale' 'SWISU' 'Blueste']
Condition1 ['Norm' 'Feedr' 'PosN' 'Artery' 'RRAe' 'RRNn' 'RRAn' 'PosA' 'RRNe']
Condition2 ['Norm' 'Artery' 'RRNn' 'Feedr' 'PosN' 'PosA' 'RRAn' 'RRAe']
BldgType ['1Fam' '2fmCon' 'Duplex' 'TwnhsE' 'Twnhs']
HouseStyle ['2Story' '1Story' '1.5Fin' '1.5Unf' 'SFoyer' 'SLvl' '2.5Unf' '2.5Fin']
RoofStyle ['Gable' 'Hip' 'Gambrel' 'Mansard' 'Flat' 'Shed']
RoofMatl ['CompShg' 'WdShngl' 'Metal' 'WdShake' 'Membran' 'Tar&Grv' 'Roll'
 'ClyTile']
Exterior1st ['VinylSd' 'MetalSd' 'Wd Sdng' 'HdBoard' 'BrkFace' 'WdShing' 'CemntBd'
 'Plywood' 'AsbShng' 'Stucco' 'BrkComm' 'AsphShn' 'Stone' 'ImStucc'
 'CBlock']
Exterior2nd ['VinylSd' 'MetalSd' 'Wd Shng' 'HdBoard' 'Plywood' 'Wd Sdng' 'CmentBd'
 'BrkFace' 'Stucco' 'AsbShng' 'Brk Cmn' 'ImStucc' 'AsphShn' 'Stone'
 'Other' 'CBlock']
MasVnrType ['BrkFace' 'None' 'Stone' 'BrkCmn' nan]
ExterQual ['Gd' 'TA' 'Ex' 'Fa']
ExterCond ['TA' 'Gd' 'Fa' 'Po' 'Ex']
Foundation ['PConc' 'CBlock' 'BrkTil' 'Wood' 'Slab' 'Stone']
BsmtQual ['Gd' 'TA' 'Ex' nan 'Fa']
BsmtCond ['TA' 'Gd' nan 'Fa' 'Po']
BsmtExposure ['No' 'Gd' 'Mn' 'Av' nan]
BsmtFinType1 ['GLQ' 'ALQ' 'Unf' 'Rec' 'BLQ' nan 'LwQ']
BsmtFinType2 ['Unf' 'BLQ' nan 'ALQ' 'Rec' 'LwQ' 'GLQ']
Heating ['GasA' 'GasW' 'Grav' 'Wall' 'OthW' 'Floor']
HeatingQC ['Ex' 'Gd' 'TA' 'Fa' 'Po']
CentralAir ['Y' 'N']
Electrical ['SBrkr' 'FuseF' 'FuseA' 'FuseP' 'Mix' nan]
KitchenQual ['Gd' 'TA' 'Ex' 'Fa']
Functional ['Typ' 'Min1' 'Maj1' 'Min2' 'Mod' 'Maj2' 'Sev']
FireplaceQu [nan 'TA' 'Gd' 'Fa' 'Ex' 'Po']
GarageType ['Attchd' 'Detchd' 'BuiltIn' 'CarPort' nan 'Basment' '2Types']
GarageFinish ['RFn' 'Unf' 'Fin' nan]
GarageQual ['TA' 'Fa' 'Gd' nan 'Ex' 'Po']
GarageCond ['TA' 'Fa' nan 'Gd' 'Po' 'Ex']
PavedDrive ['Y' 'N' 'P']
PoolQC [nan 'Ex' 'Fa' 'Gd']
Fence [nan 'MnPrv' 'GdWo' 'GdPrv' 'MnWw']
MiscFeature [nan 'Shed' 'Gar2' 'Othr' 'TenC']
SaleType ['WD' 'New' 'COD' 'ConLD' 'ConLI' 'CWD' 'ConLw' 'Con' 'Oth']
SaleCondition ['Normal' 'Abnorml' 'Partial' 'AdjLand' 'Alloca' 'Family']
```

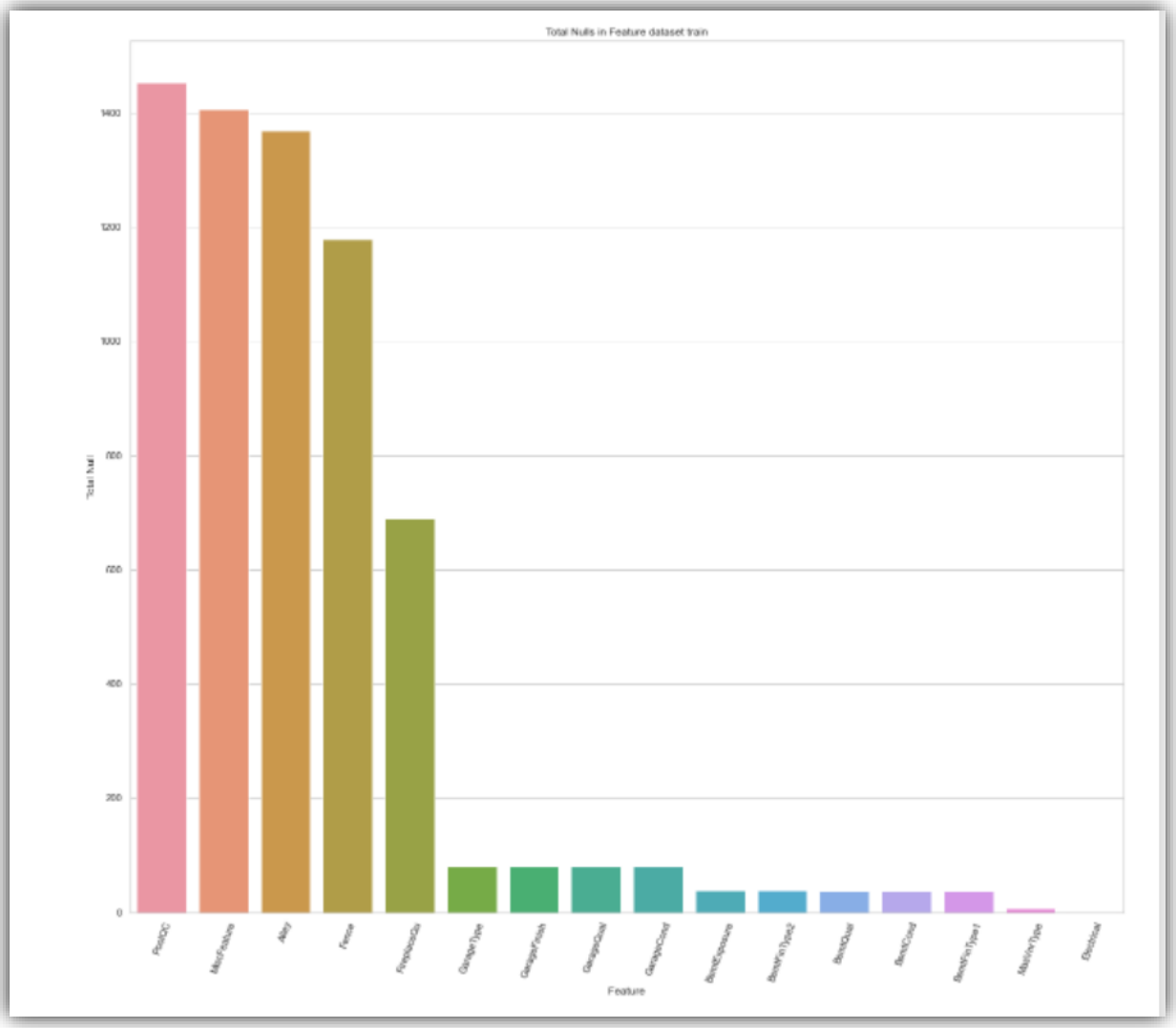
```
# count value
for k in df_train_cat:
    print(df_train_cat[k].value_counts())
```

```
RL         1151
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64
Pave    1454
Grvl       6
Name: Street, dtype: int64
Grvl    50
Pave    41
Name: Alley, dtype: int64
Reg    925
IR1    484
IR2     41
IR3     10
Name: LotShape, dtype: int64
Lvl    1311
Bnk      63
```

3-2-2 Affichage des valeurs nulles dans train dataset

```
null_list = []
for col in df_train_cat.columns:
    null = df_train_cat[col].isnull().sum()
    if null != 0 :
        null_list.append([col,null])
null_df = pd.DataFrame(null_list,columns=['Feature','Null'])
null_df.set_index('Feature')
null_df['Total Null'] = null_df['Null']
print("Total columns with null:")
print(len(null_df))
print("Total null values:")
print(null_df['Total Null'].sum(axis=0))
sns.set_palette(sns.color_palette("tab10"))
sns.barplot(data=null_df.sort_values(by='Total Null',ascending = False).head(19), x='Feature',y='Total Null')
plt.xticks(rotation = 70)
plt.title("Total Nulls in Feature dataset train")
plt.show()
```

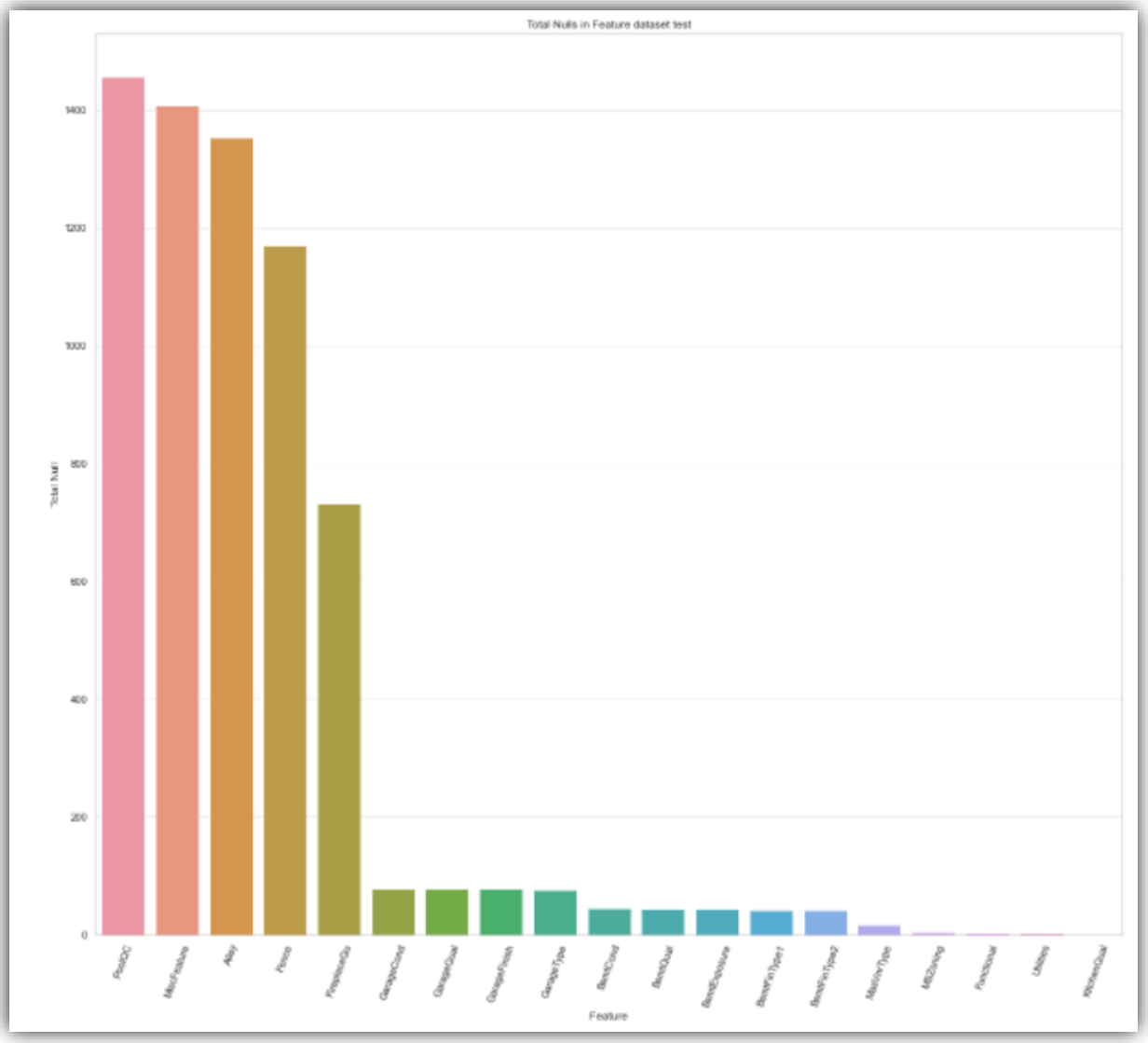
```
Total columns with null:
16
Total null values:
6617
```



3-2-3 Affichage des valeurs nulles dans test dataset

```
null_list = []
for col in df_test_cat.columns:
    null = df_test_cat[col].isnull().sum()
    if null != 0 :
        null_list.append([col,null])
null_df = pd.DataFrame(null_list,columns=['Feature','Null'])
null_df.set_index('Feature')
null_df['Total Null'] = null_df['Null']
print("Total columns with null:")
print(len(null_df))
print("Total null values:")
print(null_df['Total Null'].sum(axis=0))
sns.set_palette(sns.color_palette("tab10"))
sns.barplot(data=null_df.sort_values(by='Total Null',ascending = False).head(19), x='Feature',y='Total Null')
plt.xticks(rotation = 70)
plt.title("Total Nulls in Feature dataset test")
plt.show()
```

```
Total columns with null:
22
Total null values:
6670
```



3-2-4 Suppression des colonnes contenant plus de 200 valeurs nulles dans train et test dataset

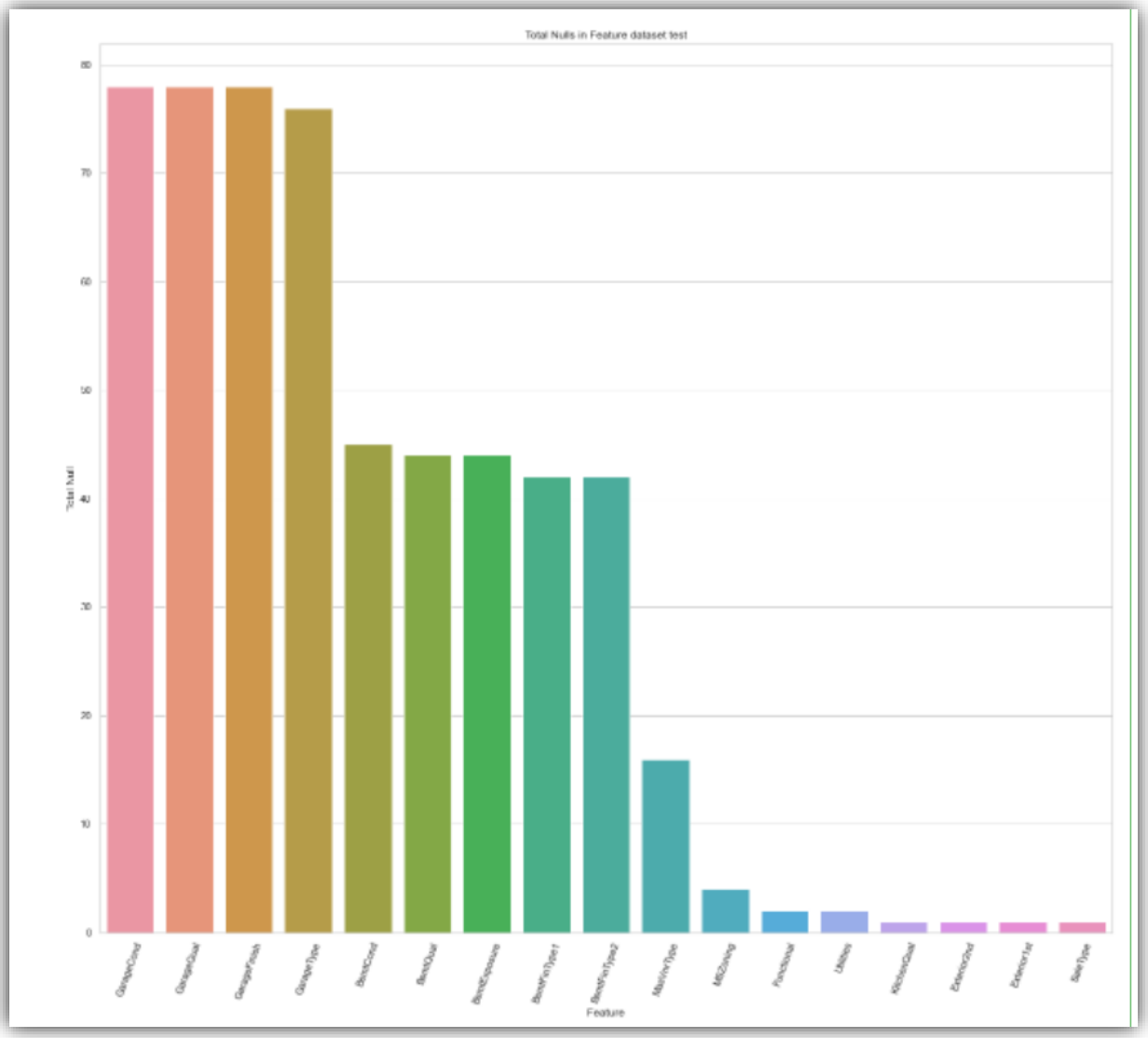
```
col_with_null=[]
for col in df_train_cat.columns:

    if df_train_cat[col].isnull().sum()> 150:
        col_with_null.append(col)
        df_train_cat.drop([col], axis = 1, inplace = True)
        df_test_cat.drop([col], axis = 1, inplace = True)
```

```
C:\Users\Dell\anaconda3\lib\site-packages\pandas\core\frame.py:4906: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

3-2- 5 Affichage de test et train dataset après la suppression des nuls

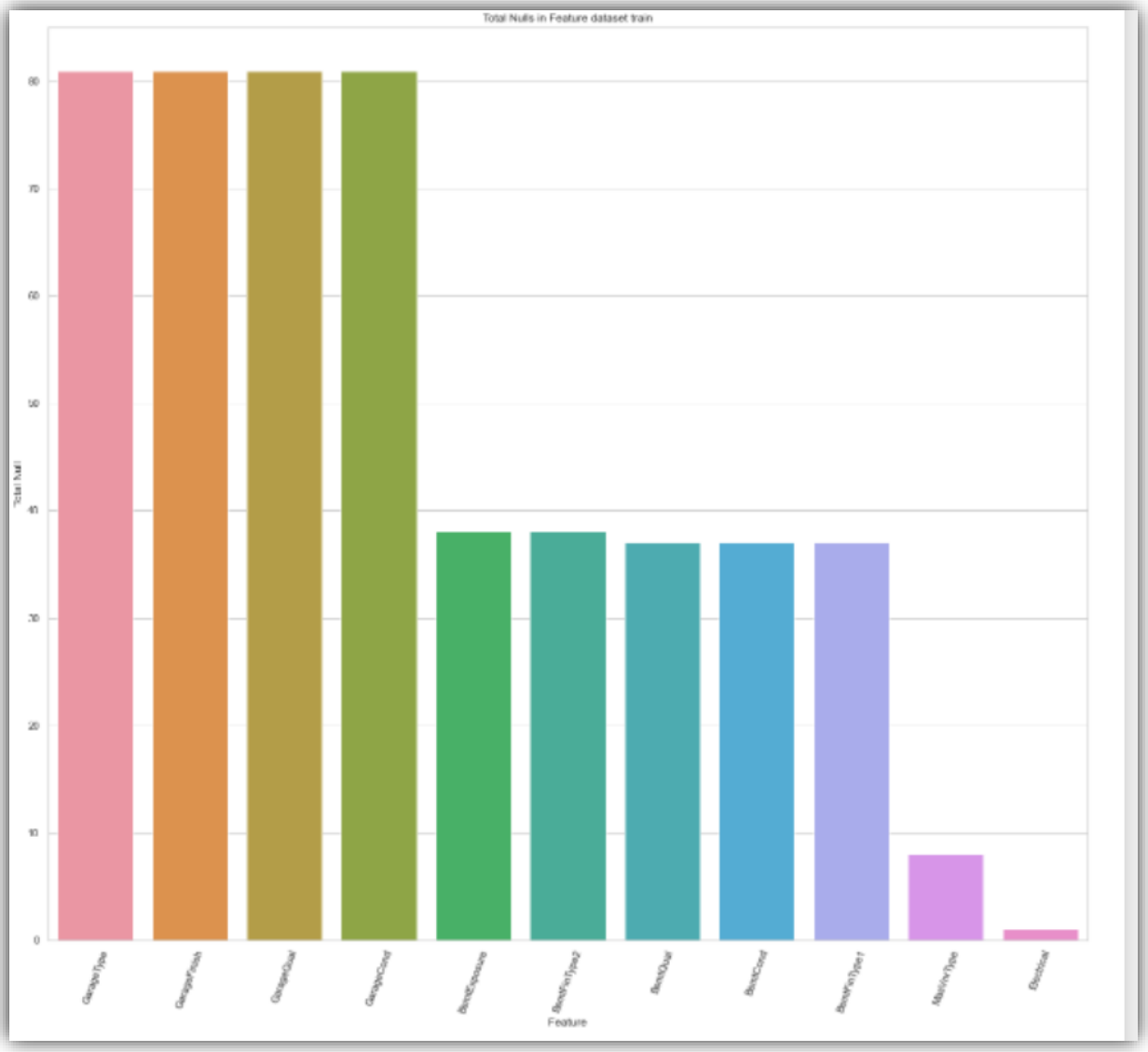
```
null_list = []
for col in df_test_cat.columns:
    null = df_test_cat[col].isnull().sum()
    if null != 0 :
        null_list.append([col,null])
null_df = pd.DataFrame(null_list,columns=['Feature','Null'])
null_df.set_index('Feature')
null_df['Total Null'] = null_df['Null']
print("Total columns with null:")
print(len(null_df))
print("Total null values:")
print(null_df['Total Null'].sum(axis=0))
sns.set_palette(sns.color_palette("tab10"))
sns.barplot(data=null_df.sort_values(by='Total Null',ascending = False).head(19), x='Feature',y='Total Null')
plt.xticks(rotation = 70)
plt.title("Total Nulls in Feature dataset test")
plt.show()
```

```
Total columns with null:
17
Total null values:
555
```



```
null_list = []
for col in df_train_cat.columns:
    null = df_train_cat[col].isnull().sum()
    if null != 0 :
        null_list.append([col,null])
null_df = pd.DataFrame(null_list,columns=['Feature','Null'])
null_df.set_index('Feature')
null_df['Total Null'] = null_df['Null']
print("Total columns with null:")
print(len(null_df))
print("Total null values:")
print(null_df['Total Null'].sum(axis=0))
sns.set_palette(sns.color_palette("tab10"))
sns.barplot(data=null_df.sort_values(by='Total Null',ascending = False).head(19), x='Feature',y='Total Null')
plt.xticks(rotation = 70)
plt.title("Total Nulls in Feature dataset train")
plt.show()
```

```
Total columns with null:
11
Total null values:
520
```



Imputation des valeurs manquantes in test and train dataset

```
# Imputation of missing values (NaNs) with SimpleImputer
my_imputer = SimpleImputer(strategy="most_frequent")
#train
df_train_imputed_cat = pd.DataFrame(my_imputer.fit_transform(df_train_cat))
df_train_imputed_cat.columns = df_train_cat.columns
# test
df_test_imputed_cat = pd.DataFrame(my_imputer.fit_transform(df_test_cat))
df_test_imputed_cat.columns = df_test_cat.columns
```

```
for k in df_train_imputed_cat:
    print(df_train_imputed_cat[k].value_counts())
```

```
RL         1151
RM          218
FV           65
RH           16
C (all)      10
Name: MSZoning, dtype: int64
Pave    1454
Grvl       6
Name: Street, dtype: int64
Reg    925
IR1    484
IR2     41
IR3     10
Name: LotShape, dtype: int64
Lvl    1311
Bnk      63
HLS      50
Low      36
Name: LandContour, dtype: int64
```

Après nettoyage:

```
# train
null_list = []

for col in df_train_imputed_cat.columns:
    null = df_train_imputed_cat[col].isnull().sum()
    if null != 0 :
        null_list.append([col,null])

null_df = pd.DataFrame(null_list,columns=['Feature','Null'])
null_df.set_index('Feature')
null_df['Total Null'] = null_df['Null']



print("Total columns with null:")
print(len(null_df))
print("Total null values:")
print(null_df['Total Null'].sum(axis=0))
print("Total columns:")
df_train_imputed_cat.shape
```

```
Total columns with null:
0
Total null values:
0
Total columns:

(1460, 38)
```

```
# test
null_list = []

for col in df_test_imputed_cat.columns:
    null = df_test_imputed_cat[col].isnull().sum()
    if null != 0 :
        null_list.append([col,null])

null_df = pd.DataFrame(null_list,columns=['Feature','Null'])
null_df.set_index('Feature')
null_df['Total Null'] = null_df['Null']



print("Total columns with null:")
print(len(null_df))
print("Total null values:")
print(null_df['Total Null'].sum(axis=0))
print("Total columns:")
df_test_imputed_cat.shape
```

```
Total columns with null:
0
Total null values:
0
Total columns:

(1459, 38)
```

3-2- 6 encodage :

```
for c in df_train_imputed_cat:
    lbl = LabelEncoder()
    lbl.fit(list(df_train_imputed_cat[c].values))
    df_train_imputed_cat[c] = lbl.transform(list(df_train_imputed_cat[c].values))

for c in df_test_imputed_cat:
    lbl.fit(list(df_test_imputed_cat[c].values))
    df_test_imputed_cat[c] = lbl.transform(list(df_test_imputed_cat[c].values))
```

```
df_train_imputed_cat.head()
```

	MSZoning	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	Condition1	Condition2	...	Electrical	KitchenQual	Functional	G
0	3	1	3	3	0	4	0	5	2	2	...	4	2	6	
1	3	1	3	3	0	2	0	24	1	2	...	4	3	6	
2	3	1	0	3	0	4	0	5	2	2	...	4	2	6	
3	3	1	0	3	0	0	0	6	2	2	...	4	2	6	
4	3	1	0	3	0	2	0	15	2	2	...	4	2	6	

5 rows × 38 columns

```
Entrée [424]: df_train_imputed_cat.dtypes

Out[424]: MSZoning         int32
          Street           int32
          LotShape         int32
          LandContour      int32
          Utilities        int32
          LotConfig        int32
          LandSlope        int32
          Neighborhood     int32
          Condition1       int32
          Condition2       int32
          BldgType         int32
          HouseStyle       int32
          RoofStyle        int32
          RoofMatl         int32
          Exterior1st      int32
          Exterior2nd      int32
          MasVnrType       int32
          ExterQual        int32
          ExterCond        int32
          Foundation       int32
          BsmtQual         int32
          BsmtCond         int32
          BsmtExposure     int32
          BsmtFinType1     int32
          BsmtFinType2     int32
          Heating          int32
          HeatingQC        int32
          CentralAir       int32
          Electrical       int32
          KitchenQual      int32
          Functional       int32
          GarageType       int32
          GarageFinish     int32
          GarageQual       int32
          GarageCond       int32
          PavedDrive       int32
          SaleType         int32
          SaleCondition    int32
          dtype: object
```

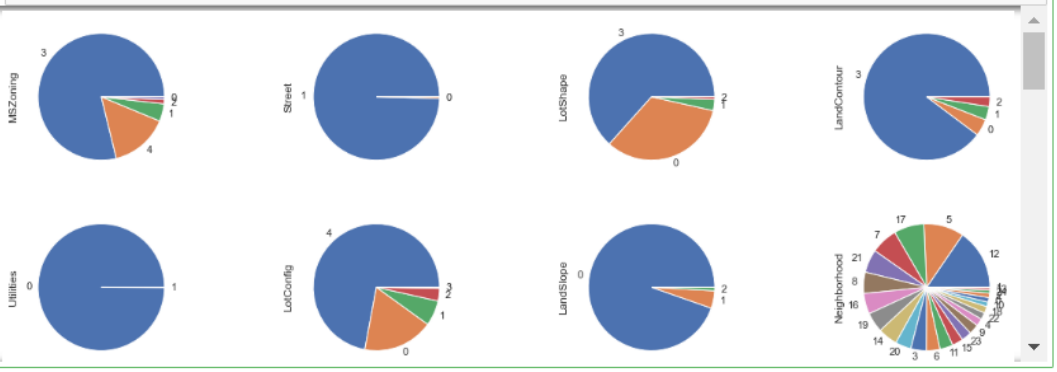
4- Concatenation des categorical features et numreical features

```
# Add binary features to numreical features
# Train set
df_train_new = df_train_imputed.join(df_train_imputed_cat)
print(f"Train set: {df_train_new.shape}")

# Test set
df_test_new = pd.DataFrame(df_test_imputed).join(df_test_imputed_cat)
print(f"Test set: {df_test_new.shape}")
```

```
Train set: (1460, 55)
Test set: (1459, 54)
```

4-1 suppression des caractéristiques quasi-constantes où 70 % des valeurs sont similaires ou constantes

Entrée [426]:
```
sns.set_theme()
fig=plt.figure(figsize=(20,40))
for i in range(len((df_train_new.select_dtypes(include='int')).columns)):
    fig.add_subplot(11,4,i+1)
    df_train_new.select_dtypes(include='int').iloc[:,i].value_counts().plot(kind="pie", subplots=True)
```

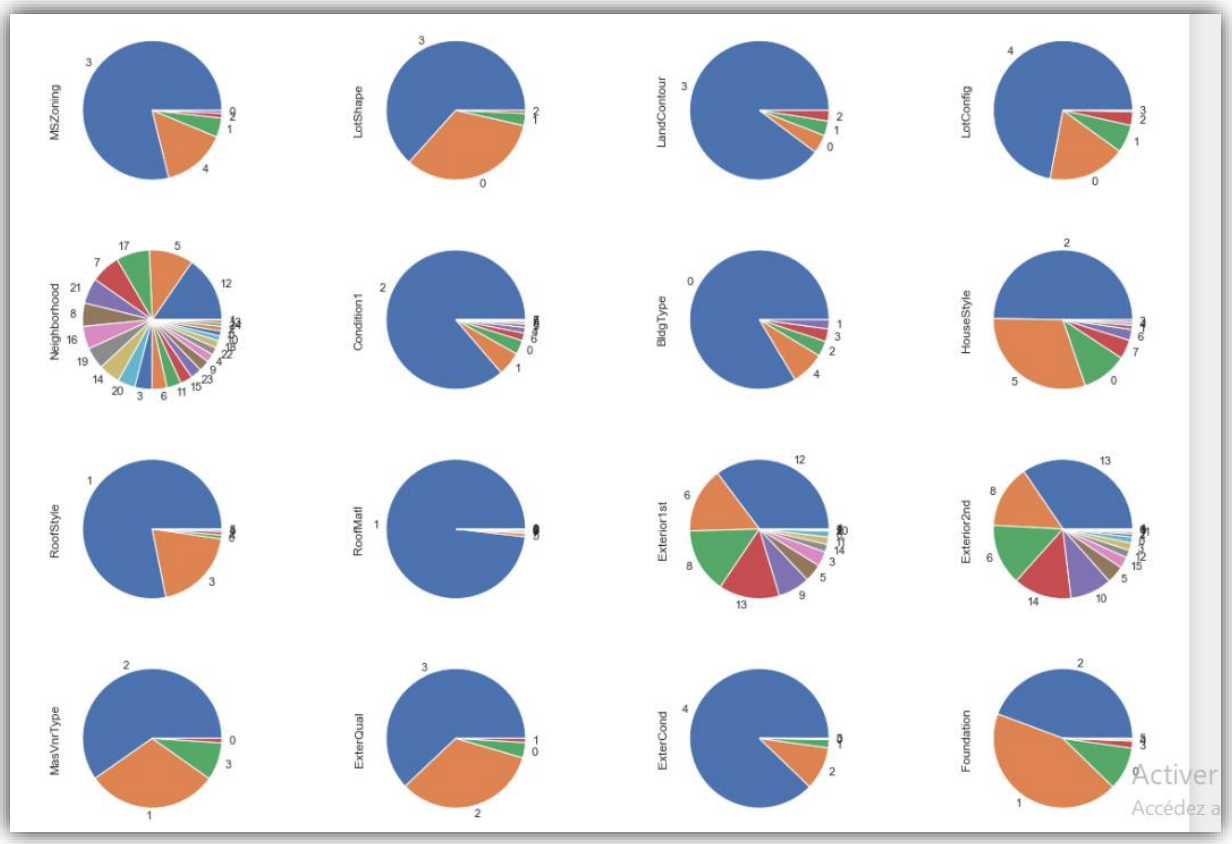


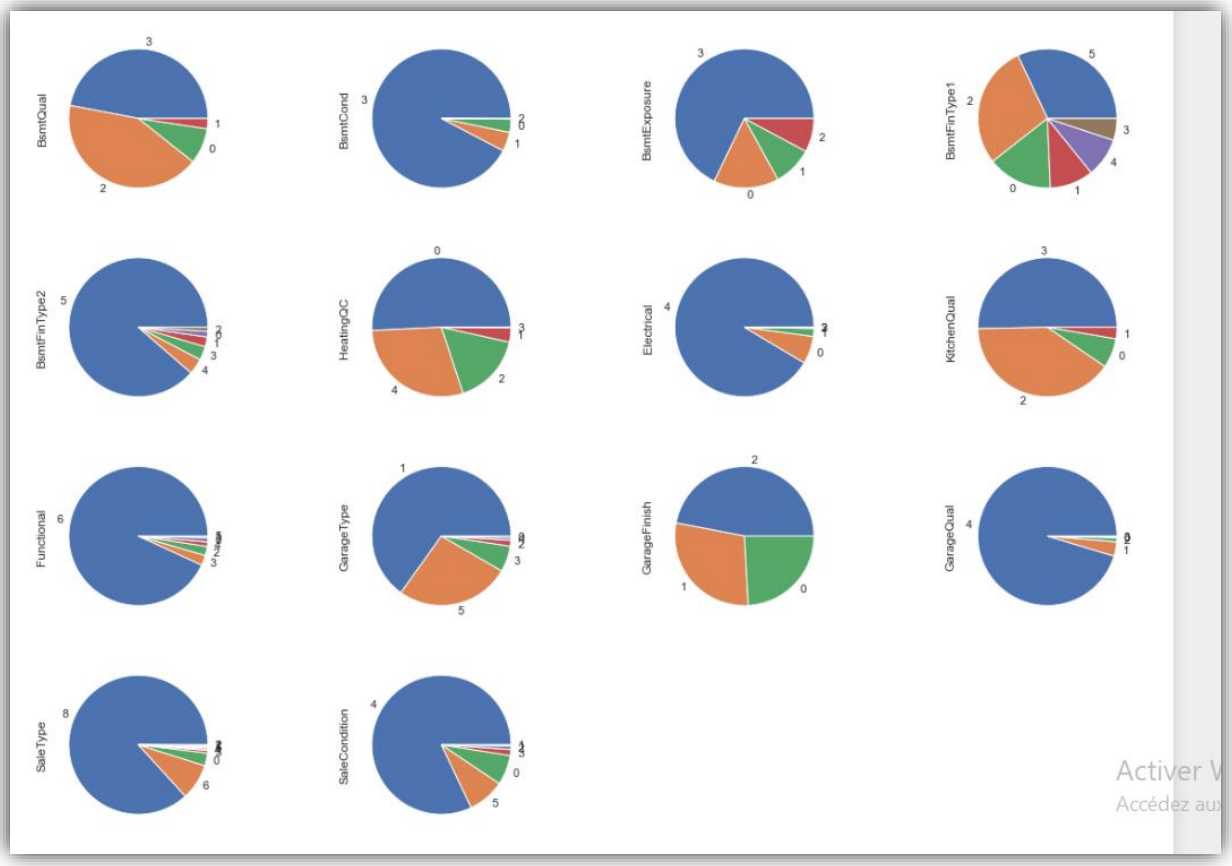
Entrée [427]:
```
# Let's drop quasi-constant features where 70% of the values are similar or constant
sel = VarianceThreshold(threshold=0.3) # 0.05: drop column where 70% of the values are constant

# fit finds the features with constant variance
sel.fit(df_train_new.iloc[:, :-1])


# Get the number of features that are not constant
print(f"Number of retained features: {sum(sel.get_support())}")
print(
    f"\nNumber of quasi_constant features: {len(df_train_new.iloc[:, :-1].columns) - sum(sel.get_support())}")

quasi_constant_features_list = [x for x in df_train_new.iloc[:, :-
                                                             1].columns if x not in df_train_new.iloc[:, :-1].columns[sel.get_s

print(
    f"\nQuasi-constant features to be dropped: {quasi_constant_features_list}")


# Let's drop these columns from df_train_new and df_train_new
df_train_new.drop(quasi_constant_features_list, axis=1, inplace=True)
df_test_new.drop(quasi_constant_features_list, axis=1, inplace=True)
```

```
Number of retained features: 46

Number of quasi_constant features: 8

Quasi-constant features to be dropped: ['Street', 'Utilities', 'LandSlope', 'Condition2', 'Heating', 'CentralAir', 'GarageCon
d', 'PavedDrive']
```

Après la suppression :

```
sns.set_theme()
fig=plt.figure(figsize=(20,40))
for i in range(len((df_train_new.select_dtypes(include='int')).columns)):
    fig.add_subplot(11,4,i+1)
    df_train_new.select_dtypes(include='int').iloc[:,i].value_counts().plot(kind="pie", subplots=True)
```





4-2 concatenation de training dataset et test dataset

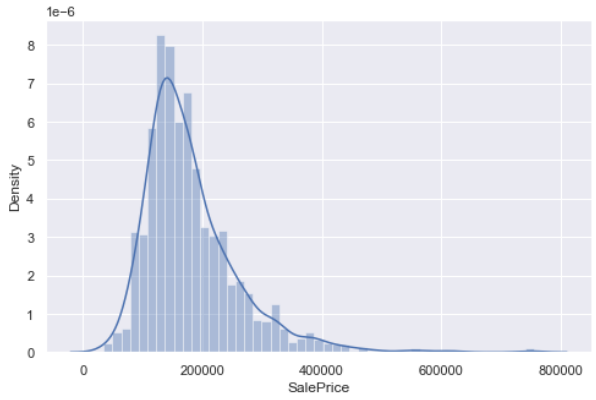
```
data = pd.concat([df_train_new,df_test_new])
data.isnull().sum().sum()
```

```
48163
```

4-3-Inspecter la fonction cible (SalePrice)

```
30]: plt.figure(figsize=(8,5))
     sns.distplot(df_train_new["SalePrice"])
     plt.show()
```

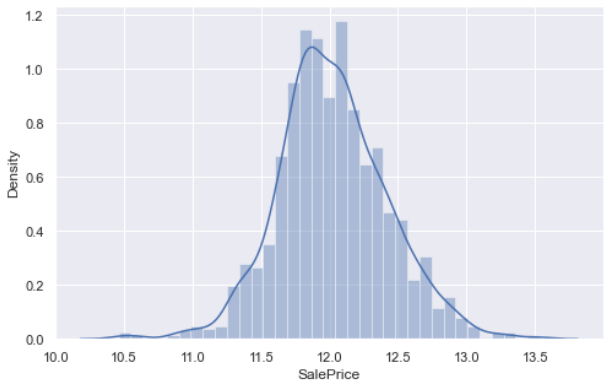
```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function
will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar fl
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Nous pouvons voir que la distribution de la variable cible est biaisée. De nombreux modèles linéaires supposent que les données sont normalement distribuées. Une distribution asymétrique rendrait également difficile l'interprétation des coefficients du modèle. Et cela conduit souvent à des problèmes tels que le sur ajustement. Pour éviter de tels problèmes plus tard dans le processus, nous distribuerons normalement notre variable cible.

```
df_train_new["SalePrice"]=df_train_new["SalePrice"].apply(np.log)
plt.figure(figsize=(8,5))
sns.distplot(df_train_new["SalePrice"])
plt.show()
```

```
C:\Users\Dell\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated function and
will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



NOTE :

Nous devrions appliquer expo après la modélisation. La distribution de l'entité cible semble normale maintenant. Nous sommes donc bons pour aller plus loin.

5- Préparation des données pour la modélisation

```
# Extract the features (X) and the target (y)
# Features (X)
X = df_train_new[[i for i in list(
    df_train_new.columns) if i != "SalePrice"]]
print(X.shape)

# Target (y)
y = df_train_new.loc[:, "SalePrice"]
print(y.shape)
print(y)
```

```
(1460, 46)
(1460,)
0       12.25
1       12.11
2       12.32
3       11.85
4       12.43
        ...
1455    12.07
1456    12.25
1457    12.49
1458    11.86
1459    11.90
Name: SalePrice, Length: 1460, dtype: float64
```

```
# split data to test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, shuffle=True)
print(f"X_train:{X_train.shape}\ny_train:{y_train.shape}")
print(f"\nX_test:{X_test.shape}\ny_test:{y_test.shape}")
```

```
X_train:(1168, 46)
y_train:(1168,)

X_test:(292, 46)
y_test:(292,)
```

```
# list ereur and score and algorithm
model_list = []
MAE_list = []
MSE_list = []
RMSE_list = []
r2_score_list = []
```

6- Modélisation

6.1 Random forestRegressor

random forest regressor est un méta estimateur qui ajuste un certain nombre d'arbres de décision de classification sur divers sous-échantillons de l'ensemble de données et utilise la moyenne pour améliorer la précision prédictive et contrôler le surajustement. La taille du sous-échantillon est contrôlée avec le paramètre **max_samples** si **bootstrap=True** (par défaut), sinon l'ensemble de données entier est utilisé pour construire chaque arbre.

```
RandomForest = RandomForestRegressor(n_estimators = 400,min_samples_split = 2,min_samples_leaf = 1,max_features= 'sqrt',max_dept
model_list.append(RandomForest.__class__.__name__)
RandomForest.fit(X_train, y_train)
predictions = RandomForest.predict(X_test)
#predictions
```

```
predictions.shape
```

```
(292,)
```

```
y_test.shape
```

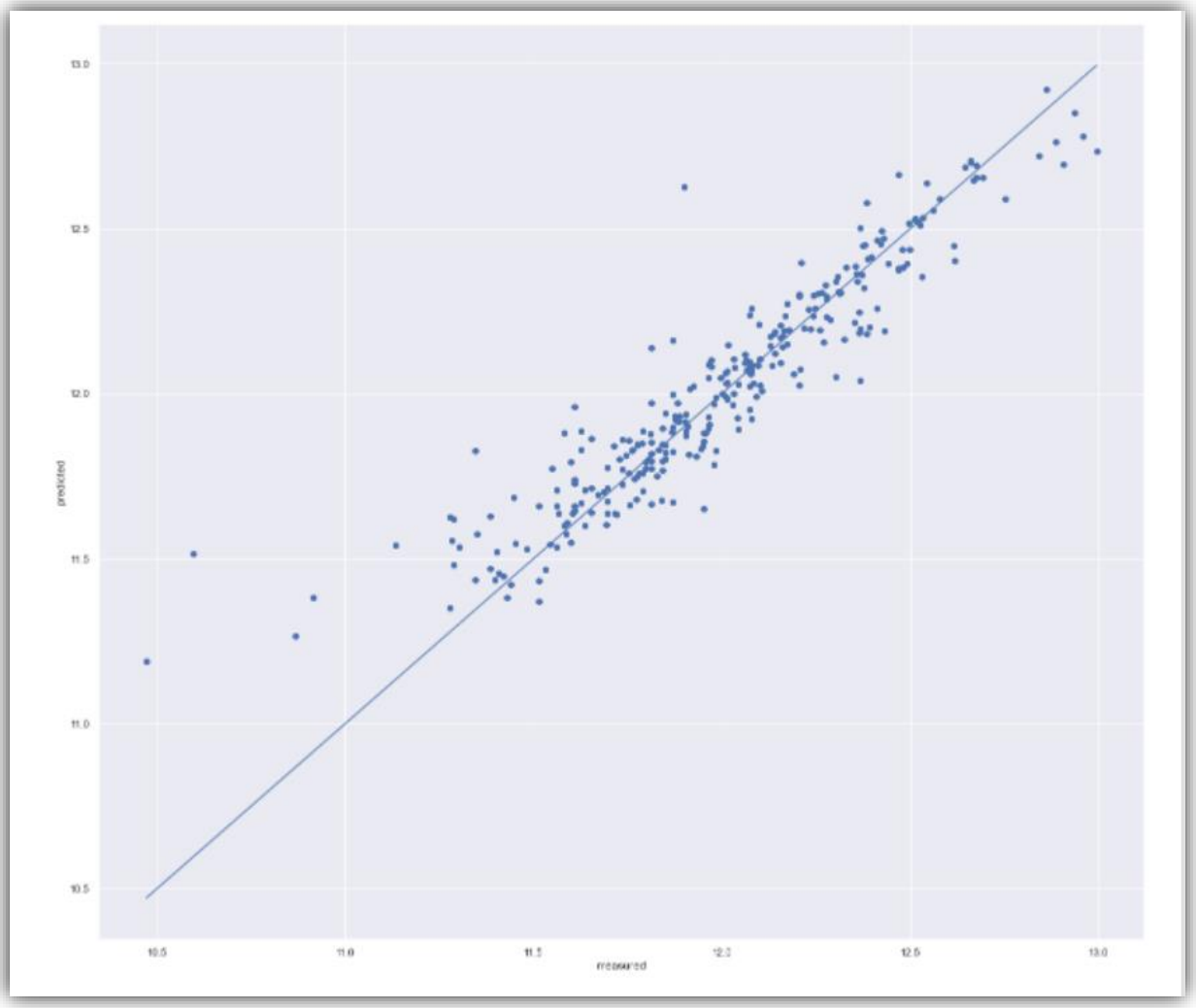
```
(292,)
```

❖ **Score de Random forestRegressor**

```
print("R2 score", r2_score(y_test, predictions))
r2_score_list.append(r2_score(y_test, predictions))
```

```
R2 score 0.8644410182432066
```

```
fig, ax = plt.subplots()
ax.scatter(y_test, predictions)
ax.plot([y_test.min(),
         y_test.max()],
        [y_test.min(),y_test.max()])
ax.set_xlabel('measured')
ax.set_ylabel('predicted')
plt.show()
```




```
from sklearn import metrics
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predictions)))
MAE_list.append(metrics.mean_absolute_error(y_test, predictions))
MSE_list.append(metrics.mean_squared_error(y_test, predictions))
RMSE_list.append(np.sqrt(metrics.mean_squared_error(y_test, predictions)))
```

```
Mean Absolute Error: 0.09231568918824558
Mean Squared Error: 0.02078195654054397
Root Mean Squared Error: 0.1441594830059541
```

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': predictions})
df.head(10)
```

	Actual	Predicted
423	12.66	12.70
1407	11.63	11.67
360	11.96	11.88
837	11.51	11.43
393	11.51	11.37
1199	11.90	11.89
679	11.76	11.83
611	11.90	11.94
964	12.28	12.30
36	11.88	11.91

6.2 Regression linear:

La régression linéaire simple est un type d'analyse de régression où le nombre de variables indépendantes est égal à un et il existe une relation linéaire entre la variable indépendante (x) et dépendante (y). La ligne rouge dans le graphique ci-dessus est appelée la ligne droite la mieux ajustée. Sur la base des points de données donnés, nous essayons de tracer une ligne qui modélise le mieux les points. La ligne peut être modélisée sur la base de l'équation linéaire ci-dessous.

Équation linéaire

$$y = a_0 + a_1 * x$$

Le motif de l'algorithme de régression linéaire est de trouver les meilleures valeurs pour a_0 et a_1. Avant de passer à l'algorithme, examinons deux concepts importants que vous devez connaître pour mieux comprendre la régression linéaire

```
# regression linear
regressor = LinearRegression()
model_list.append(regressor.__class__.__name__)
regressor.fit(X_train, y_train) #training the algorithm
#To retrieve the intercept:
print(regressor.intercept_)
#For retrieving the slope:
print(regressor.coef_)
```

```
11.996745389933391
[ 0.11480177  0.03096827  0.04469467  0.00492103  0.03311564  0.06018292
  0.00180406  0.02721568  0.05350393  0.00386185 -0.00064147  0.01063139
  0.02547697  0.03432148  0.01195278 -0.00099994 -0.01818628 -0.00972817
  0.00744759 -0.00106573  0.0008087  -0.00081958 -0.02450919 -0.00961695
  0.00889738  0.02691349 -0.00548264  0.00244195 -0.00399359  0.0144347
  0.00115621  0.00144539 -0.01915319  0.01495445 -0.01526198 -0.01648871
 -0.00904066 -0.00951113  0.00533332 -0.02085203  0.0222642  -0.00522324
 -0.00136287  0.01541592 -0.00175661  0.02126839]
```

```
#prediction
y_pred = regressor.predict(X_test)
```

❖ Score de Regresssion linear :

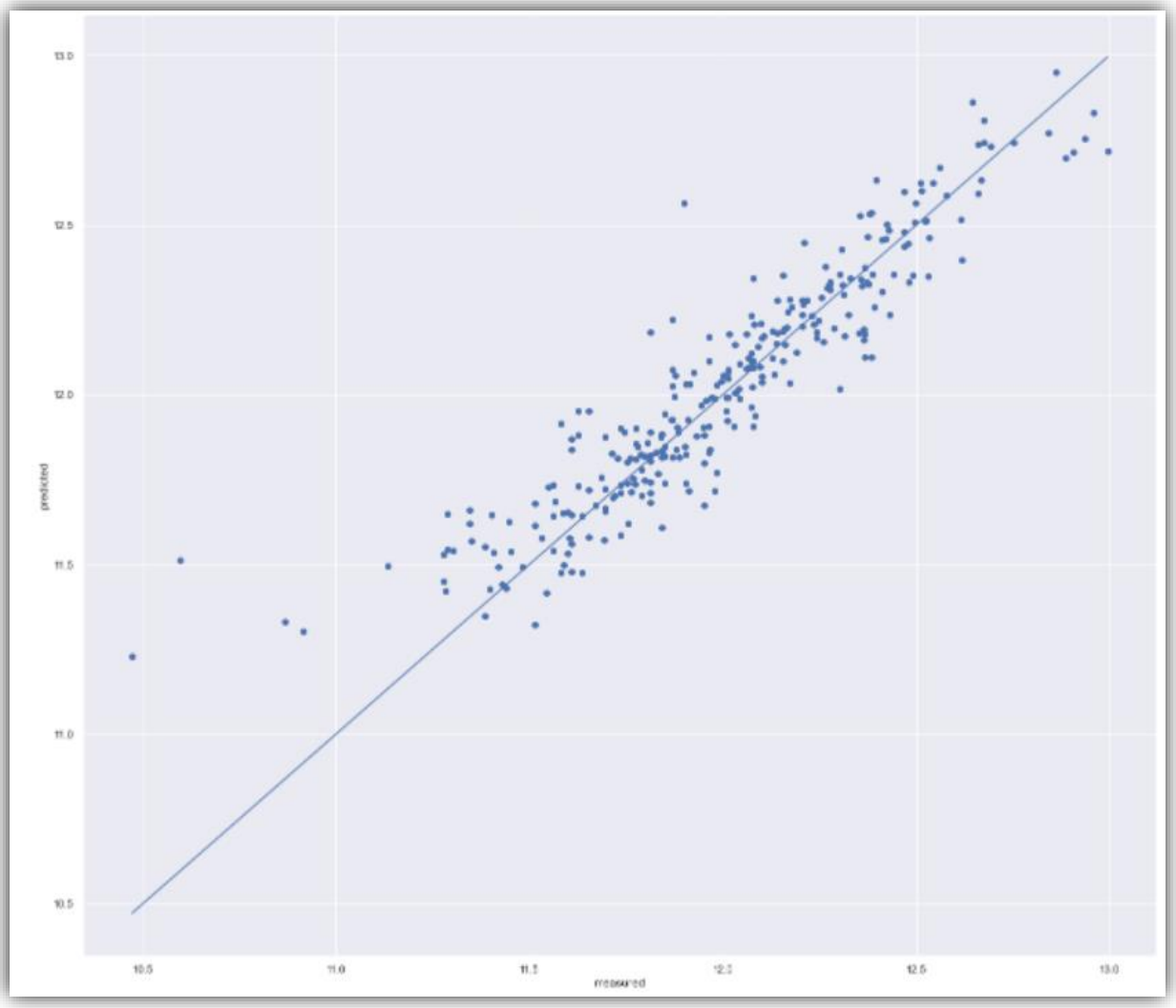
```
: # score
  print("R2 score", r2_score(y_test, y_pred))
  r2_score_list.append(r2_score(y_test, y_pred))
```

```
R2 score 0.8495998475676023
```

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df.head(20)
```

	Actual	Predicted
423	12.66	12.59
1407	11.63	11.73
360	11.96	11.98
837	11.51	11.61
393	11.51	11.32
1199	11.90	11.82
679	11.76	11.71
611	11.90	12.03
964	12.28	12.31
36	11.88	11.89
602	12.30	12.36
301	12.50	12.51
439	11.61	11.84
1081	11.80	11.75
172	12.38	12.11
252	12.06	12.08
559	12.36	12.16
796	11.87	11.99
788	11.59	11.50
118	12.68	12.81

```
# ploting
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred)
ax.plot([y_test.min(),
         y_test.max()],
        [y_test.min(),y_test.max()])
ax.set_xlabel('measured')
ax.set_ylabel('predicted')
plt.show()
```




```
# Error
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
MAE_list.append(metrics.mean_absolute_error(y_test, y_pred))
MSE_list.append(metrics.mean_squared_error(y_test, y_pred))
RMSE_list.append(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
Mean Absolute Error: 0.10483194558888667
Mean Squared Error: 0.0230571917185756
Root Mean Squared Error: 0.1518459473235147
```

6-3- Lasso Regression

La régression au lasso est une technique de régularisation. Il est utilisé sur les méthodes de régression pour une prédiction plus précise. Ce modèle utilise le retrait. Le rétrécissement est l'endroit où les valeurs des données sont rétrécies vers un point central comme moyenne. La procédure de lasso encourage les modèles simples et clairsemés (c'est-à-dire les modèles avec moins de paramètres). Ce type particulier de régression est bien adapté aux modèles présentant des niveaux élevés de multi colinéarité ou lorsque vous souhaitez automatiser certaines parties de la sélection du modèle, comme la sélection de variables/l'élimination de paramètres**.**

```
#Lasso Regression

#Initializing the Lasso Regressor
lasso_reg = Lasso(alpha=0.001)
model_list.append(lasso_reg.__class__.__name__)
#Fitting the Training data to the Lasso regressor
lasso_reg.fit(X_train,y_train)
#Predicting for X_test
y_pred_lass =lasso_reg.predict(X_test)
```

❖ Score de lasso Regressor

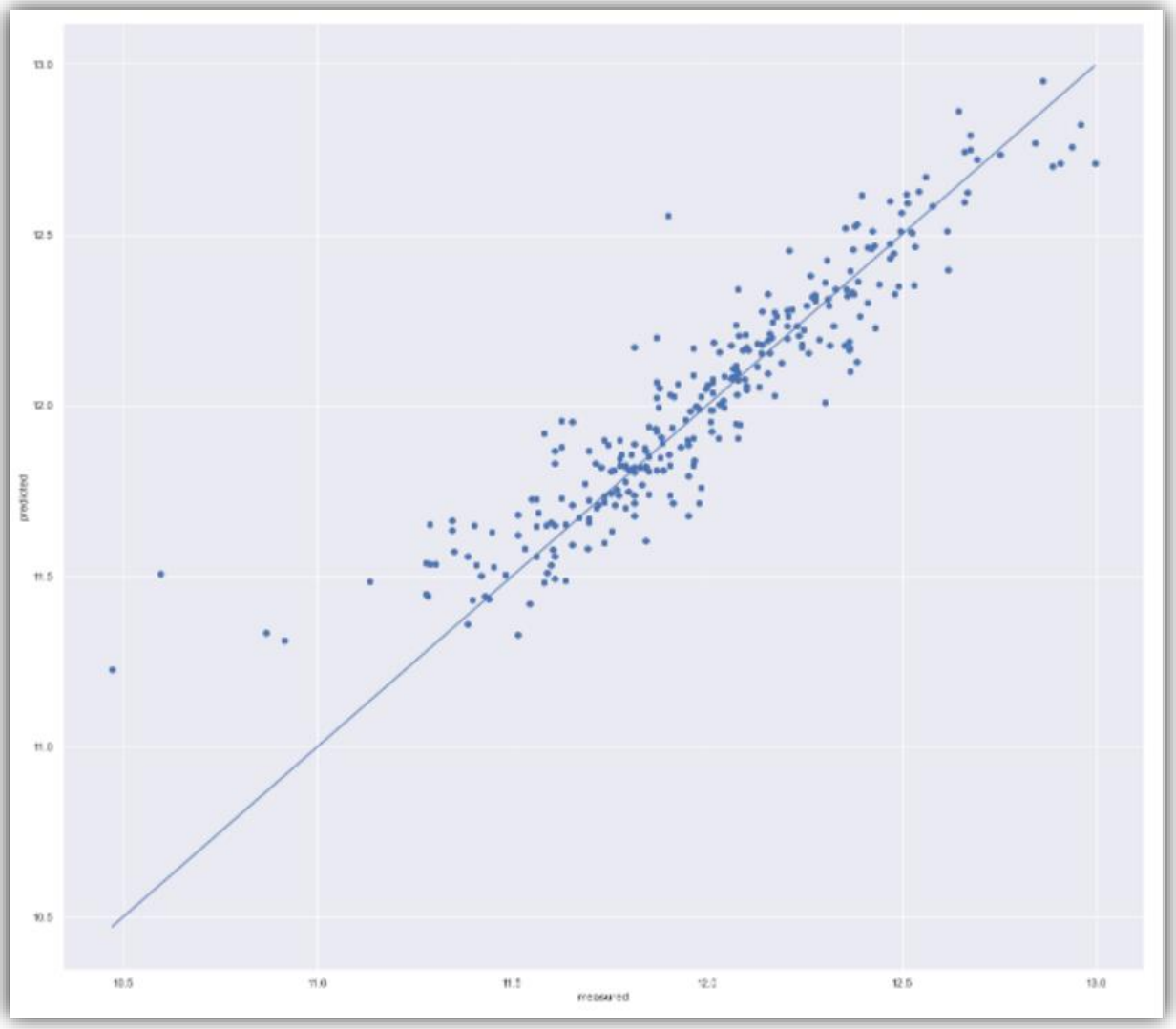
```
print("R2 score", r2_score(y_test, y_pred_lass))
r2_score_list.append(r2_score(y_test, y_pred_lass))
```

```
R2 score 0.8509778938626734
```

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred_lass})
df.head(10)
```

	Actual	Predicted
423	12.66	12.60
1407	11.63	11.73
360	11.96	11.98
837	11.51	11.62
393	11.51	11.33
1199	11.90	11.82
679	11.76	11.71
611	11.90	12.03
964	12.28	12.31
36	11.88	11.89

```
#plotting
fig, ax = plt.subplots()
ax.scatter(y_test, y_pred_lass)
ax.plot([y_test.min(),
         y_test.max()],
        [y_test.min(),y_test.max()])
ax.set_xlabel('measured')
ax.set_ylabel('predicted')
plt.show()
```




```
# error
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_lass))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_lass))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred_lass)))
MAE_list.append(metrics.mean_absolute_error(y_test, y_pred_lass))
MSE_list.append(metrics.mean_squared_error(y_test, y_pred_lass))
RMSE_list.append(np.sqrt(metrics.mean_squared_error(y_test, y_pred_lass)))
```

```
Mean Absolute Error: 0.10454065522312181
Mean Squared Error: 0.022845929448500384
Root Mean Squared Error: 0.1511486997909687
```

6-4- XGB Regressor

```
#XGB
xgbr = xgb.XGBRegressor()
model_list.append(xgbr.__class__.__name__)
xgbr.fit(X_train, y_train)
#Predicting for X_test
predict_xgb = xgbr.predict(X_test)
```

❖ Score de XGB Regressor

```
print("R2 score", r2_score(y_test, predict_xgb))
r2_score_list.append(r2_score(y_test,predict_xgb))
```

```
R2 score 0.860995958787952
```

```
#error
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predict_xgb))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predict_xgb))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predict_xgb)))
MAE_list.append(metrics.mean_absolute_error(y_test, predict_xgb))
MSE_list.append(metrics.mean_squared_error(y_test, predict_xgb))
RMSE_list.append(np.sqrt(metrics.mean_squared_error(y_test, predict_xgb)))
```

```
Mean Absolute Error: 0.09801434425068288
Mean Squared Error: 0.021310103587319083
Root Mean Squared Error: 0.1459798054092383
```

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': predict_xgb})
df.head(10)
```

	Actual	Predicted
423	12.66	12.75
1407	11.63	11.70
360	11.96	11.93
837	11.51	11.45
393	11.51	11.38
1199	11.90	11.75
679	11.76	11.84
611	11.90	11.89
964	12.28	12.26
36	11.88	11.91

6-5- SVR : Support Vector Regression

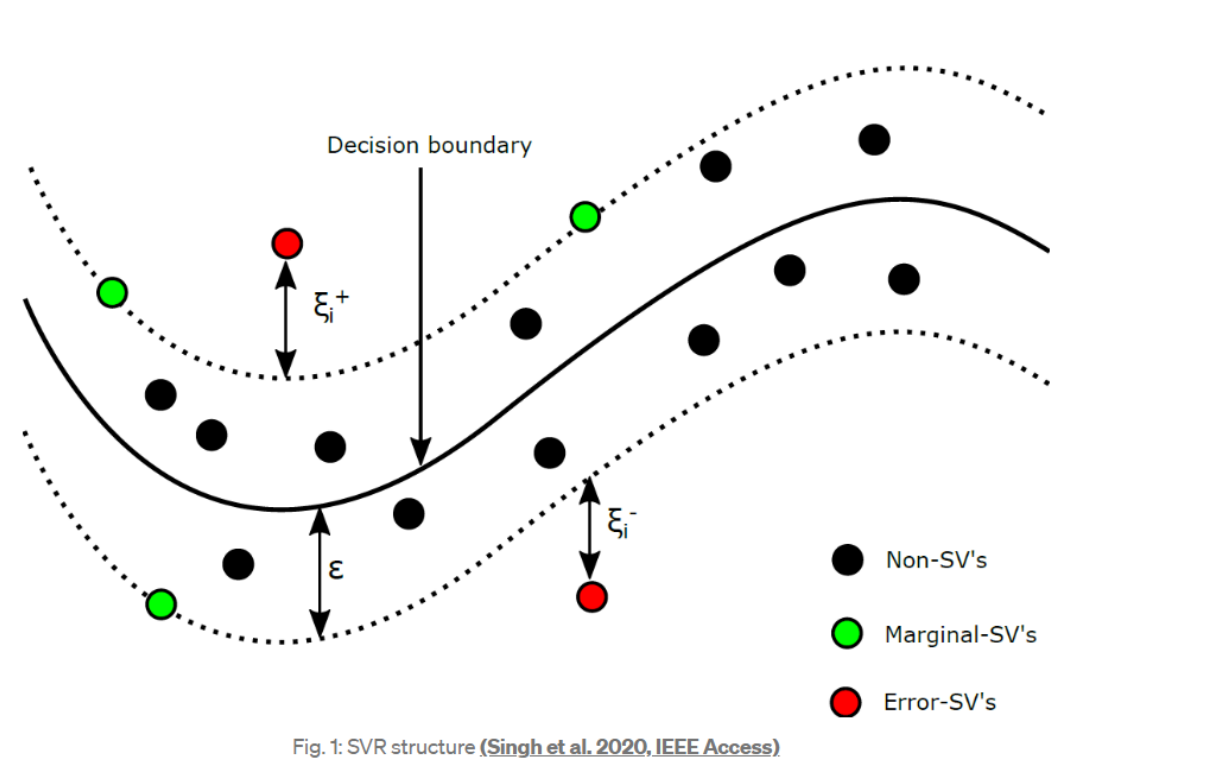


Fig. 1: SVR structure **(Singh et al. 2020, IEEE Access)**

SVR a été initialement proposé par Drucker et al., qui est une technique d'apprentissage supervisé, basée sur le concept des vecteurs de support de Vapnik. Le SVR vise à réduire l'erreur en déterminant l'hyperplan et en minimisant l'écart entre les valeurs prédites et observées. La minimisation de la valeur de w dans l'équation ci-dessous est similaire à la valeur définie pour maximiser la marge, comme illustré à la Fig. 1

```
regressor = SVR(kernel = 'rbf')
model_list.append(regressor.__class__.__name__)
regressor.fit(X_train, y_train)
```

```
SVR()
```

```
predict_SVR = regressor.predict(X_test)
```

❖ Score de SVR

```
print("R2 score", r2_score(y_test, predict_SVR))
r2_score_list.append(r2_score(y_test, predict_SVR))
```

```
R2 score 0.8613427519811511
```

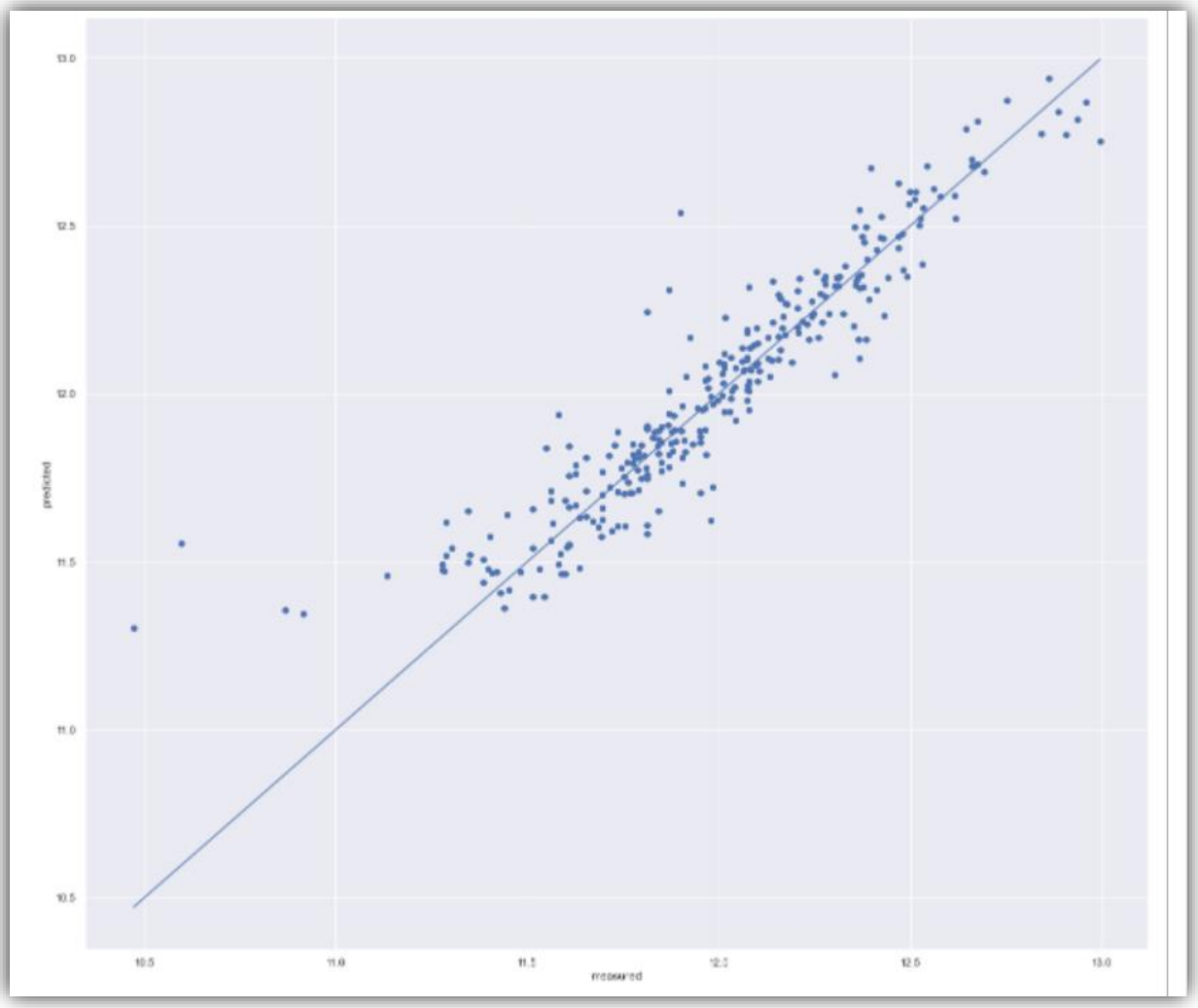
```
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, predict_SVR))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, predict_SVR))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, predict_SVR)))
MAE_list.append(metrics.mean_absolute_error(y_test, predict_SVR))
MSE_list.append(metrics.mean_squared_error(y_test, predict_SVR))
RMSE_list.append(np.sqrt(metrics.mean_squared_error(y_test, predict_SVR)))
```

```
Mean Absolute Error: 0.09494086148692205
Mean Squared Error: 0.021256938234671702
Root Mean Squared Error: 0.14579759337750298
```

```
df = pd.DataFrame({'Actual': y_test, 'Predicted': predict_SVR})
df.head(10)
```

	Actual	Predicted
423	12.66	12.68
1407	11.63	11.67
360	11.96	11.95
837	11.51	11.54
393	11.51	11.40
1199	11.90	11.81
679	11.76	11.74
611	11.90	11.96
964	12.28	12.29
36	11.88	11.89

```
# ploting
fig, ax = plt.subplots()
ax.scatter(y_test, predict_SVR)
ax.plot([y_test.min(),
         y_test.max()],
        [y_test.min(),y_test.max()])
ax.set_xlabel('measured')
ax.set_ylabel('predicted')
plt.show()
```


13.0
12.5
12.0
11.5
11.0
10.5
predicted
10.5
11.0
11.5
12.0
12.5
13.0

```
RMSE_list
```

```
[0.1441594830059541,
 0.1518459473235147,
 0.1511486997909687,
 0.1459798054092383,
 0.14579759337750298]
```

```
MSE_list
```

```
[0.02078195654054397,
 0.0230571917185756,
 0.022845929448500384,
 0.021310103587319083,
 0.021256938234671702]
```

```
model_list
```

```
['RandomForestRegressor', 'LinearRegression', 'Lasso', 'XGBRegressor', 'SVR']
```

```
MAE_list
```

```
[0.09231568918824558,
 0.10483194558888667,
 0.10454065522312181,
 0.09801434425068288,
 0.09494086148692205]
```

```
r2_score_list
```

```
[0.8644410182432066,
 0.8495998475676023,
 0.8509778938626734,
 0.860995958787952,
 0.8613427519811511]
```

7- choix du meilleur model

```
# Create a table with pd.DataFrame for all results
model_results = pd.DataFrame({"Model": model_list,
                              "MAE": MAE_list,
                              "MSE": MSE_list,
                              "RMSE": RMSE_list,
                              "R2_score":r2_score_list})

model_results
```

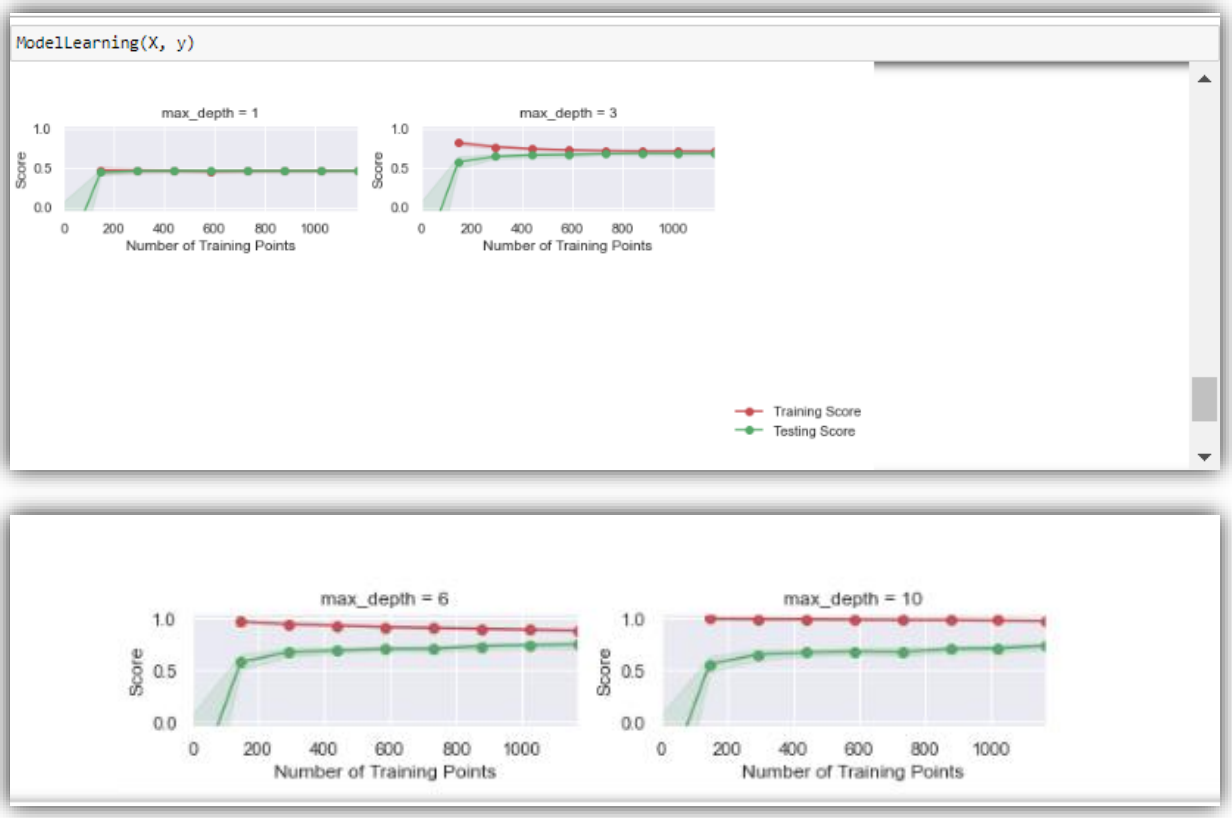
	Model	MAE	MSE	RMSE	R2_score
0	RandomForestRegressor	0.09	0.02	0.14	0.86
1	LinearRegression	0.10	0.02	0.15	0.85
2	Lasso	0.10	0.02	0.15	0.85
3	XGBRegressor	0.10	0.02	0.15	0.86
4	SVR	0.09	0.02	0.15	0.86

Remarque:

Nous choisirons SVR car l'erreur est petite et R2_score est le plus grand

8- Verification de l’existance de overfiting ou underfiting:

```
def ModelLearning(X, y):
    """ Calculates the performance of several models with varying sizes of training data.
        The learning and testing scores for each model are then plotted. """

    # Create 10 cross-validation sets for training and testing
    cv = ShuffleSplit(n_splits = 10, test_size = 0.2, random_state = 0)

    # Generate the training set sizes increasing by 50
    train_sizes = np.rint(np.linspace(1, X.shape[0]*0.8 - 1, 9)).astype(int)

    # Create the figure window
    fig = plt.figure(figsize=(10,7))

    # Create three different models based on max_depth
    for k, depth in enumerate([1,3,6,10]):

        # Create a  DecisionTreeRegressor
        regressor = DecisionTreeRegressor(max_depth = depth)

        # Calculate the training and testing scores
        sizes, train_scores, test_scores = learning_curve(regressor, X, y, \
            cv = cv, train_sizes = train_sizes, scoring = 'r2')

        # Find the mean and standard deviation for smoothing
        train_std = np.std(train_scores, axis = 1)
        train_mean = np.mean(train_scores, axis = 1)
        test_std = np.std(test_scores, axis = 1)
        test_mean = np.mean(test_scores, axis = 1)

        # Subplot the learning curve
        ax = fig.add_subplot(2, 2, k+1)
        ax.plot(sizes, train_mean, 'o-', color = 'r', label = 'Training Score')
        ax.plot(sizes, test_mean, 'o-', color = 'g', label = 'Testing Score')
        ax.fill_between(sizes, train_mean - train_std, \
            train_mean + train_std, alpha = 0.15, color = 'r')
        ax.fill_between(sizes, test_mean - test_std, \
            test_mean + test_std, alpha = 0.15, color = 'g')

        # Labels
        ax.set_title('max_depth = %s'%(depth))
        ax.set_xlabel('Number of Training Points')
        ax.set_ylabel('Score')
        ax.set_xlim([0, X.shape[0]*0.8])
        ax.set_ylim([-0.05, 1.05])

    # Visual aesthetics
    ax.legend(bbox_to_anchor=(1.05, 2.05), loc='lower left', borderaxespad = 0.)
    fig.suptitle('Decision Tree Regressor Learning Performances', fontsize = 16, y = 1.03)
    fig.tight_layout()
    fig.show()
```



Remarque:

On Remarque qu'il n'y a pas de overfitting ou d'underfitting

9- Soumission

```
: predict_SVR.shape
(292,)

: len(Id_test_list)
1459

: df_test_new.shape
(1459, 46)
```

❖ **Application de l'expo au SalePrice**

```
#predict = regressor.predict(X_test)
# predict
# We take antilog i.e. np.exp to convert log predictions to actual prices.
predict = np.exp(regressor.predict(df_test_new))
#predict=scaler.inverse_transform(regressor.predict(X_test))
#print(regressor.predict(X_test))
```

```
submission = pd.DataFrame({"Id": Id_test_list,
                           "SalePrice": predict})
submission.head(10)
```

	Id	SalePrice
0	1461	115,493.26
1	1462	159,113.23
2	1463	182,040.07
3	1464	190,585.02
4	1465	178,832.31
5	1466	176,567.82
6	1467	171,402.04
7	1468	166,193.59
8	1469	185,930.30
9	1470	119,419.45

```
import joblib
```

```
submission.to_csv("submission.csv", index=False)
```

```
file = 'model.sav'
joblib.dump(regressor, file)
```

```
['model.sav']
```

References:

- https://medium.com/analytics-vidhya/support-vector-regression-svr-model-a-regression-based-machine-learning-approach-f4641670c5bb
- https://www.kaggle.com/c/house-prices-advanced-regression-techniques