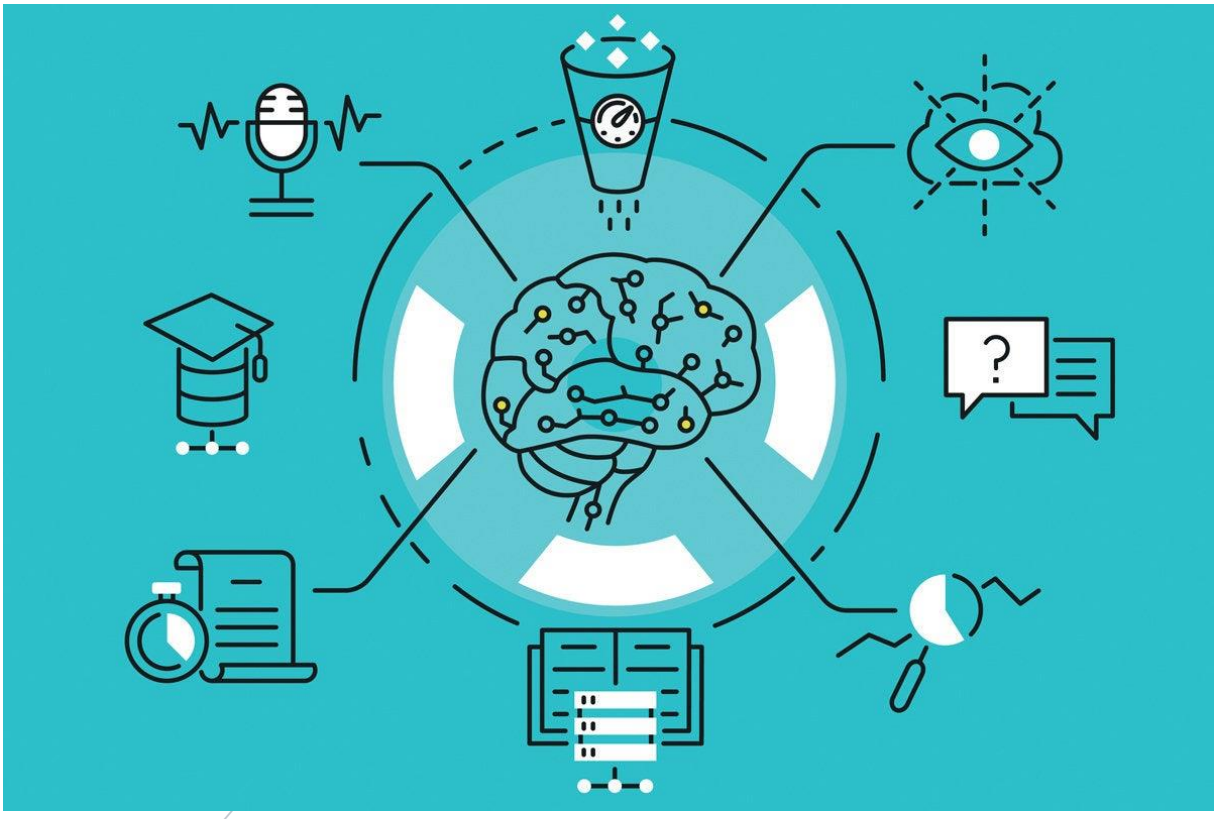
Université Abdelmalek Essaâdi
Faculté des Sciences et Techniques-Tanger
Département Génie Informatique

Machine Learning (Master SIBD)

Atelier 1 « Régression »



Réalisé Par :
Yossra safi chetouan

Encadre Par :
Pr . EL AACHAk LOTFI

Partie 1 : Régression Simple cas Expérience Salaire

1. Préparation des données

- Traiter les valeurs manquantes

```
#3.a)traiter les valeur manquantes
data.isnull().sum()
#remarque:On voit qu'il y a aucune valeurs nulles
```

```
YearsExperience    0
Salary             0
dtype: int64
```

On voit qu'il n'y a pas de valeurs manquantes

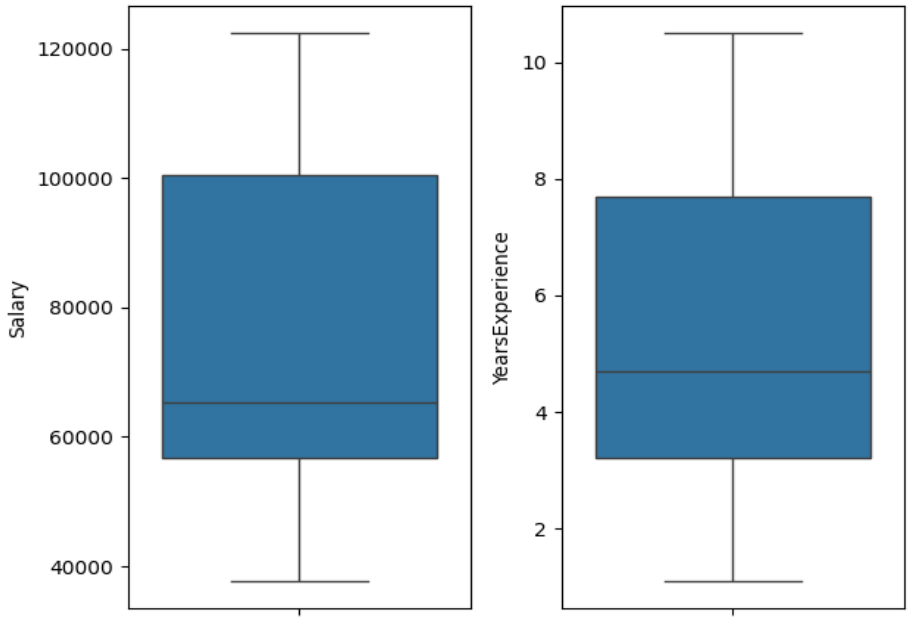
- Le traitement des valeurs dupliquées

```
#3.b)Le traitement des valeurs dupliquées
data.duplicated().sum()
#remarque:pas des valeurs dupliquées
```

```
0
```

On voit qu'il n'y a pas de valeurs dupliquées

- Traitement des valeurs aberrantes (Outliers)



On voit qu'il n'y a pas de valeurs aberrantes

2. Visualisation des données

- Afficher les dernières lignes de data set

[10]:

	YearsExperience	Salary
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

- Afficher les premières lignes

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

- Afficher les Informations sur les types de données et les valeurs manquantes

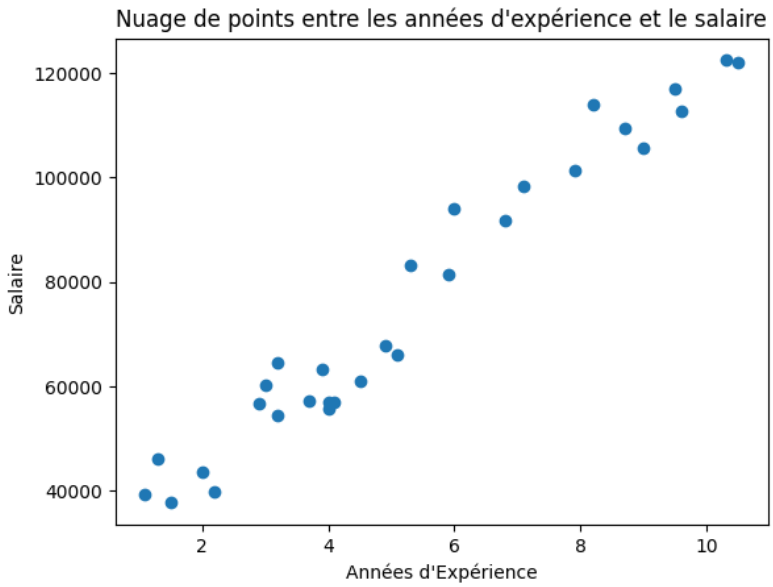
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

- Statistiques descriptives

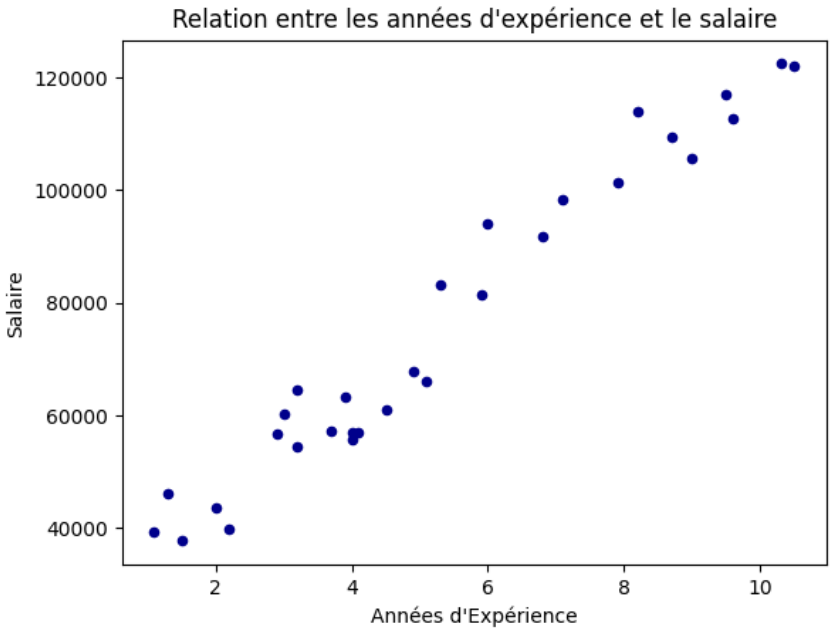
	YearsExperience	Salary
count	30.000000	30.000000
mean	5.313333	76003.000000
std	2.837888	27414.429785
min	1.100000	37731.000000
25%	3.200000	56720.750000
50%	4.700000	65237.000000
75%	7.700000	100544.750000
max	10.500000	122391.000000

- Le nombre d'observations non nulles pour les deux colonnes est de 30.
- La moyenne des salaires est de 76 003.
- La moyenne des années d'expérience est de 5,313.
- La valeur maximale dans la colonne des salaires est de 122 391.
- La valeur minimale dans la colonne des salaires est de 37 731.
- La valeur maximale dans la colonne des années d'expérience est de 10,50.
- La valeur minimale dans la colonne des années d'expérience est de 1,10.
- 25% (premier quartile): Le quartile inférieur, sous lequel se trouvent 25% des données.
- 50% (médiane ou deuxième quartile): La médiane divise les données en deux moitiés égales.
- 75% (troisième quartile): Le quartile supérieur, au-dessus duquel se trouvent 25% des données.

- Afficher la nuage des points matplotlib



- Afficher la nuage des points Pandas



3. Entraînement de modèle

- Division data en 2 data sets : traitement *08% et test 20%

```
X = data[['YearsExperience']].values
Y = data['Salary'].values
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test,Y_train, Y_test = train_test_split(X, Y, test_size=0.2,random_state=0)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(24, 1)
(6, 1)
(24,)
(6,)
```

- Entraîner le modèle

```
lmodellineaire = LinearRegression()
lmodellineaire.fit(X_train, Y_train)
```

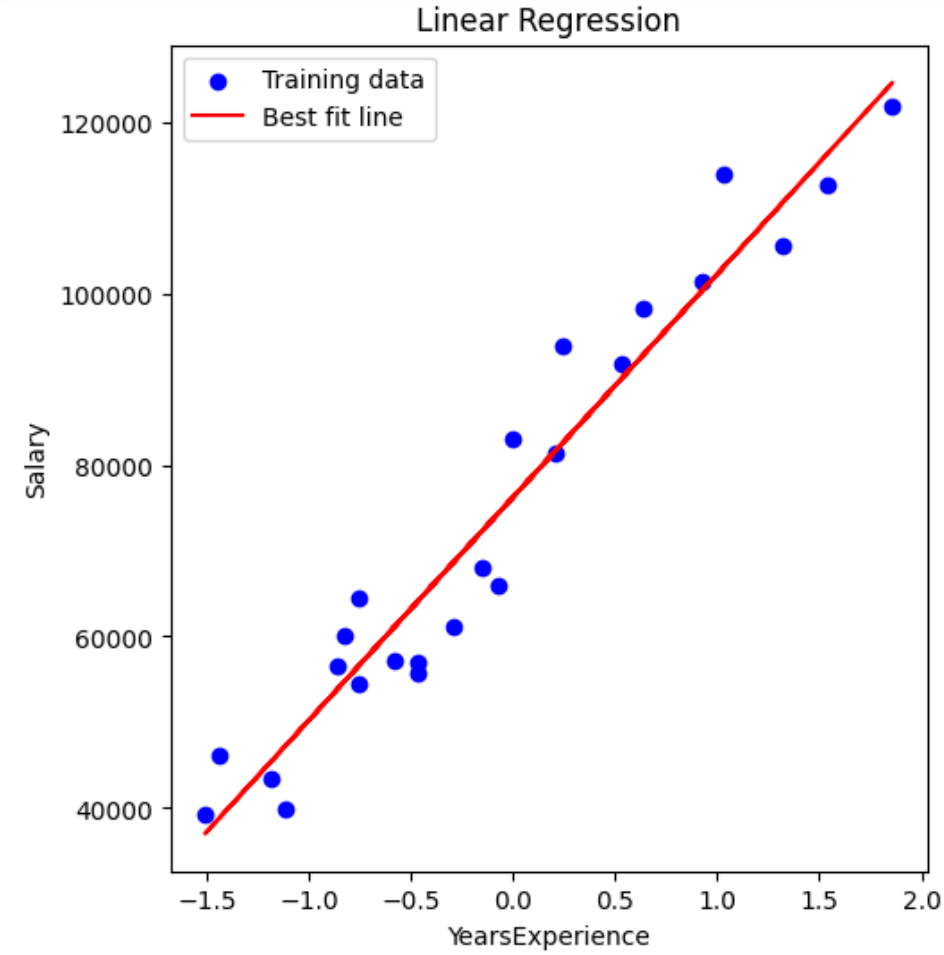
```
▾ LinearRegression
LinearRegression()
```

4. Prédiction sur les données de test

```
y_test_pred_lin = lmodellineaire.predict(X_test)
y_test_pred_lin
```

```
array([ 40748.96184072, 122699.62295594,  64961.65717022,  63099.14214487,
       115249.56285456, 107799.50275317])
```

5. Visualiser le résultat de la régression sous forme d'un graphe



6. Évaluation de modèle

```
La performance du modèle sur la base dapprentissage
Mean Squared Error (MSE): 12823412.298126549
Root Mean Squared Error (RMSE): 3580.979237321343
Mean Absolute Error (MAE): 2446.1723690465064
```

Partie 2 : Régression multiple cas d'assurance

1. Préparation des données

➢ Le traitement les valeurs manquantes

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

'Remarque: On voit qu'il y a aucune valeurs nulles '

➢ Le traitement des valeurs dupliquées

```
#3.b)Le traitement des valeurs dupliquées
print(data.duplicated().sum())
```

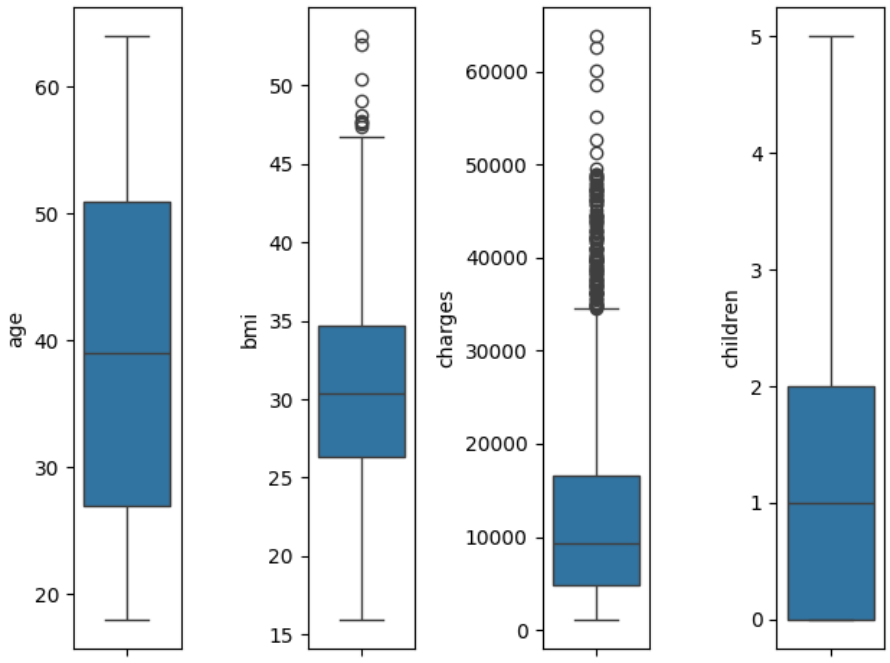
```
1
```

```
data = data.drop_duplicates()
print(data.duplicated().sum())
```

```
0
```

Il y a une seule valeur dupliquée

➢ Traitement des valeurs aberrantes (Outliers)



Il y a des valeurs aberrantes (outliers) pour les colonnes BMI et charges

Traiter valeurs aberrantes pour colonne BMI

```
Q1 = data['bmi'].quantile(0.25)
Q3 = data['bmi'].quantile(0.75)
IQR = Q3 - Q1

# Définition des bornes pour les outliers
lower_bound = Q1 - 0.4* IQR
upper_bound = Q3 + 0.4 * IQR

# Filtrage des outliers
data= data[(data['bmi'] >= lower_bound) & (data['bmi'] <= upper_bound)]

print(data)
```

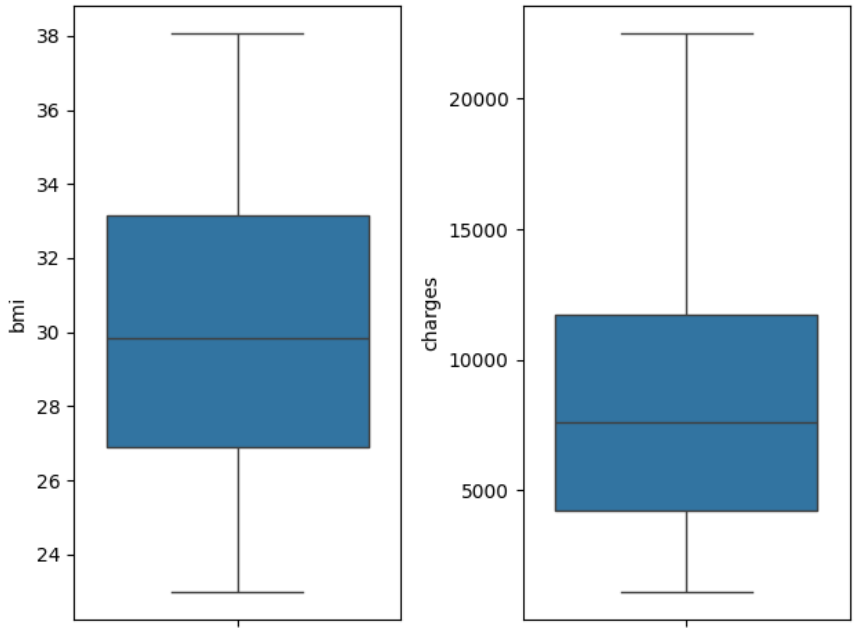
```
      age     sex    bmi  children smoker     region     charges
0      19  female  27.90         0    yes  southwest  16884.9240
1      18    male  33.77         1     no  southeast   1725.5523
2      28    male  33.00         3     no  southeast   4449.4620
4      32    male  28.88         0     no  northwest   3866.8552
5      31  female  25.74         0     no  southeast   3756.6216
...   ...     ...    ...       ...    ...        ...         ...
1333   50    male  30.97         3     no  northwest  10600.5483
1334   18  female  31.92         0     no  northeast   2205.9808
1335   18  female  36.85         0     no  southeast   1629.8335
1336   21  female  25.80         0     no  southwest   2007.9450
1337   61  female  29.07         0    yes  northwest  29141.3603

[1052 rows x 7 columns]
```

Traiter valeurs aberrantes pour colonne Charges

```
Q1 = data['charges'].quantile(0.25)
Q3 = data['charges'].quantile(0.75)
IQR = Q3 - Q1

# Définition des bornes pour les outliers
lower_bound = Q1 - 0.4 * IQR
upper_bound = Q3 + 0.4 * IQR

# Filtrage des outliers
data= data[(data['charges'] >= lower_bound) & (data['charges'] <= upper_bound)]

print(data)
```

```
      age     sex    bmi  children smoker     region      charges
0      19  female  27.90         0    yes  southwest  16884.92400
1      18    male  33.77         1     no  southeast   1725.55230
2      28    male  33.00         3     no  southeast   4449.46200
4      32    male  28.88         0     no  northwest   3866.85520
5      31  female  25.74         0     no  southeast   3756.62160
...   ...     ...    ...       ...    ...        ...          ...
1331   23  female  33.40         0     no  southwest  10795.93733
1333   50    male  30.97         3     no  northwest  10600.54830
1334   18  female  31.92         0     no  northeast   2205.98080
1335   18  female  36.85         0     no  southeast   1629.83350
1336   21  female  25.80         0     no  southwest   2007.94500

[860 rows x 7 columns]
```

Résultat :



Nous ne voyons maintenant plus de valeurs aberrantes dans les colonnes BMI et charges après traitement

- Encodage des valeurs catégoriques vers numérique

[449]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.90	0	1	3	16884.92400
1	18	1	33.77	1	0	2	1725.55230
2	28	1	33.00	3	0	2	4449.46200
4	32	1	28.88	0	0	1	3866.85520
5	31	0	25.74	0	0	2	3756.62160
...	...	...	...	...	...	...	...
1331	23	0	33.40	0	0	3	10795.93733
1333	50	1	30.97	3	0	1	10600.54830
1334	18	0	31.92	0	0	0	2205.98080
1335	18	0	36.85	0	0	2	1629.83350
1336	21	0	25.80	0	0	3	2007.94500

860 rows × 7 columns

2. Visualisation des données

- Afficher les dernières lignes de data set

	age	sex	bmi	children	smoker	region	charges
1331	23	0	33.40	0	0	3	10795.93733
1333	50	1	30.97	3	0	1	10600.54830
1334	18	0	31.92	0	0	0	2205.98080
1335	18	0	36.85	0	0	2	1629.83350
1336	21	0	25.80	0	0	3	2007.94500

- Afficher les premières lignes

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.90	0	1	3	16884.9240
1	18	1	33.77	1	0	2	1725.5523
2	28	1	33.00	3	0	2	4449.4620
4	32	1	28.88	0	0	1	3866.8552
5	31	0	25.74	0	0	2	3756.6216

- Afficher les Informations sur les types de données et les valeurs manquantes

```
<class 'pandas.core.frame.DataFrame'>
Index: 860 entries, 0 to 1336
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       860 non-null    int64  
 1   sex       860 non-null    int32  
 2   bmi       860 non-null    float64
 3   children  860 non-null    int64  
 4   smoker    860 non-null    int32  
 5   region    860 non-null    int32  
 6   charges   860 non-null    float64
dtypes: float64(2), int32(3), int64(2)
memory usage: 43.7 KB
```

- Statistiques descriptives

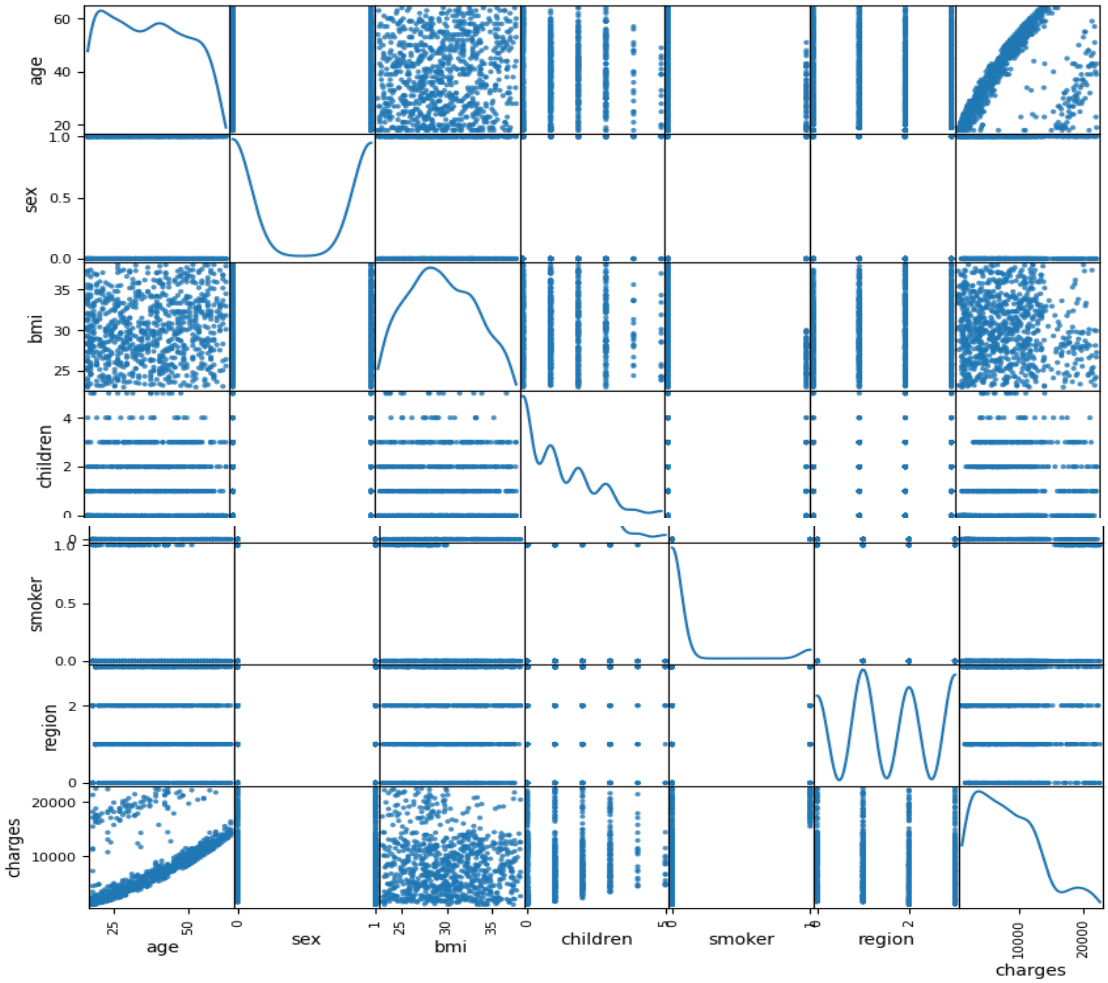
	age	sex	bmi	children	smoker	region	charges
count	860.000000	860.000000	860.000000	860.000000	860.000000	860.000000	860.000000
mean	38.639535	0.491860	30.048529	1.105814	0.073256	1.532558	8492.440043
std	13.651502	0.500225	3.950011	1.233681	0.260708	1.109850	5291.822479
min	18.000000	0.000000	22.990000	0.000000	0.000000	0.000000	1121.873900
25%	27.000000	0.000000	26.885000	0.000000	0.000000	1.000000	4236.576662
50%	39.000000	0.000000	29.830000	1.000000	0.000000	2.000000	7636.569025
75%	50.000000	1.000000	33.155000	2.000000	0.000000	3.000000	11743.457775
max	64.000000	1.000000	38.060000	5.000000	1.000000	3.000000	22493.659640

- Sélection les caractéristiques importantes en utilisant la méthode d'importance des caractéristiques (FI)

[456]:
```
X = data[['age', 'sex', 'bmi', 'children', 'smoker', 'region']]  # Features
y = data['charges']

model = ExtraTreesRegressor()
model.fit(X, y)

# Affichage de l'importance des caractéristiques
importances = model.feature_importances_
importance_dict = {X.columns[i]: imp for i, imp in enumerate(importances)}
print("Feature Importance:", importance_dict)
```

```
Feature Importance: {'age': 0.4530099831347224, 'sex': 0.01841560551585683, 'bmi': 0.1017005512145458, 'children': 0.051956331727668725, 'smoker': 0.3294
7403301043793, 'region': 0.04544349539676829}
```

- Afficher la nuage des points



3. Entraînement de modèle

- Division data en 2 data sets : traitement *08% et test 20%

```
X = data[['age','bmi','smoker']].values
Y = data['charges'].values
scaler = StandardScaler()
X = scaler.fit_transform(X)
X_train, X_test,Y_train, Y_test = train_test_split(X, Y, test_size=0.2,random_state=0)
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(688, 3)
(172, 3)
(688,)
(172,)
```

- Entraîner le modèle

```
regressor = LinearRegression()
regressor.fit(X_train, Y_train)
```

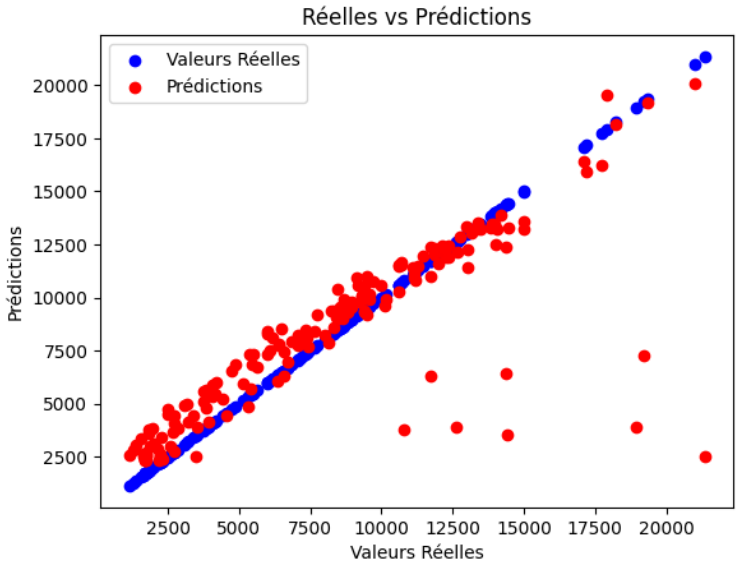
```
▾ LinearRegression
LinearRegression()
```

4. Prédiction sur les données de test

```
[464]: y_pred = regressor.predict(X_test)
       y_pred
```

```
[464]: array([ 6208.49927093, 11248.59424481, 11638.20852718,  3018.83058861,
         6224.72787829,  8249.29223103,  3721.52794628,  5473.94707874,
        11853.84888849,  8275.0693104 ,  5461.71819189,  6652.27624238,
        12726.42675086, 10290.13579444,  5770.96284298, 10455.65950645,
        11085.48892305,  7232.86954514,  8742.97900206, 10707.46705068,
        11486.01961604,  4085.96618465,  5273.3764938 ,  5626.9473374 ,
        12155.20150906,  2631.20545229,  4134.81195821, 11918.04986181,
         5952.42097878,  4778.70119091,  2646.02452056, 11821.95958133,
         4641.5447397 ,  5028.4207358 ,  4718.58328631, 10081.12939643,
         8734.33557342,  4992.00554134,  8018.57039395,  6525.26337524,
         3028.60526081,  6594.11532484,  5828.97955678,  4234.01266358,
        11260.19489178,  7200.51440351,  7923.55656507, 11728.86653219,
        11529.47928616,  6923.02901614,  7162.25772919,  7899.2637531 ,
         5049.54645598,  7193.41744957,  3903.85222154,  8668.97427706,
         6272.8358281 ,  3760.46803535,  9790.3689546 ,  8333.70145557,
        10030.3682363 , 10238.45370291, 10112.0822671 , 10395.94633812,
        11413.32616274, 14451.0080091 ,  8092.98293473,  9415.78713216,
         5216.93653871, 13231.84479823,  5546.57254866, 12975.50140199,
        10894.5733145 , 10620.28492001, 12609.76508337, 12530.64947097,
         6955.97926981,  3447.64428806, 11150.83741089, 10514.70957934,
         6092.52063526,  6473.77532491, 13834.32533123,  6868.12999535,
         5317.15066679, 11085.51651049,  6902.39460023, 11433.78512206,
        10419.79363493,  9358.516862  , 10818.67958583,  5128.29585309,
        12140.40694872,  9814.51448333,  8722.09041561, 12843.50904266,
         8602.12833038, 10542.47272525,  9542.1830759 ,  8009.83216034,
         8919.14165076,  5426.86117154,  7571.78410881, 10199.53812176,
         7920.62775579,  4909.47438693, 12221.73025629,  9374.88818105,
         7037.02615682,  7104.64575166, 13280.19064762,  3760.46803535,
        10457.05046093,  7615.88410114, 11139.49917143, 12744.81996086
```

5. Visualiser le résultat de la régression sous forme d'un graphe



6. Évaluation de modèle

```
La performance du modèle sur la base dapprentissage
Mean Absolute Error: 1348.6295945645788
Mean Squared Error: 7277241.235054895
Root Mean Squared Error: 2697.6362310465242
```

Partie 4 : Régression linière polynomial multiple cas de china GDP

1. Préparation des données

- Traitement les valeurs manquantes

```
#3.a)traiter les valeur manquantes
data.isnull().sum()
```

```
Year     0
Value    0
dtype: int64
```

On voit qu'il n'y a pas de valeurs manquantes

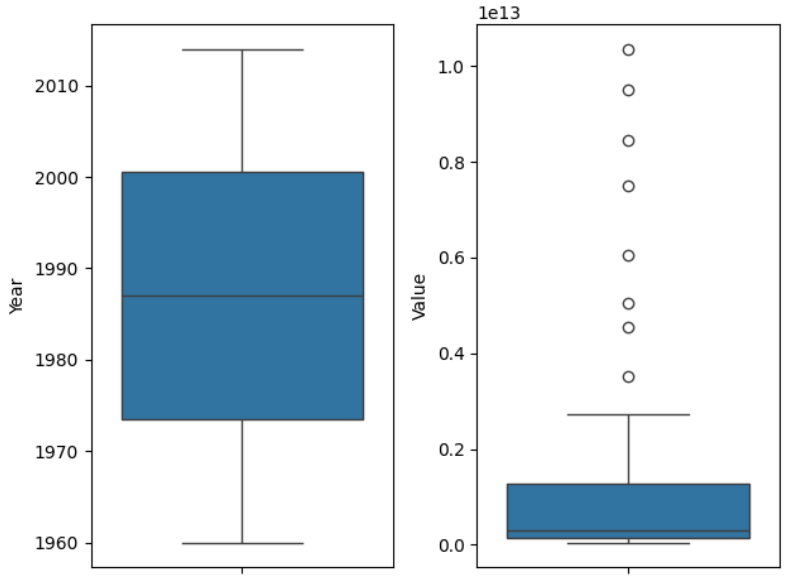
- Traitement des valeurs dupliquées

```
#3.b)Le traitement des valeurs dupliquées
data.duplicated().sum()
```

```
0
```

On voit qu'il n'y a pas de valeurs dupliquées

- Traitement des valeurs aberrantes (Outliers)



Il y a des valeurs aberrantes dans la colonne Value

```
Q1 = data['Value'].quantile(0.25)
Q3 = data['Value'].quantile(0.75)
IQR = Q3 - Q1

# Définition des bornes pour les outliers
lower_bound = Q1 -3 * IQR
upper_bound = Q3 + 3* IQR

# Filtrage des outliers
data= data[(data['Value'] >= lower_bound) & (data['Value'] <= upper_bound)]
data
```

	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10
5	1965	6.970915e+10
6	1966	7.587943e+10
7	1967	7.205703e+10
8	1968	6.999350e+10

2. Visualisation des données

- Afficher les dernières lignes de data set

	Year	Value
44	2004	1.941746e+12
45	2005	2.268599e+12
46	2006	2.729784e+12
47	2007	3.523094e+12
48	2008	4.558431e+12

- Afficher les premières lignes

	Year	Value
0	1960	5.918412e+10
1	1961	4.955705e+10
2	1962	4.668518e+10
3	1963	5.009730e+10
4	1964	5.906225e+10

- Afficher les Informations sur les types de données et les valeurs manquantes

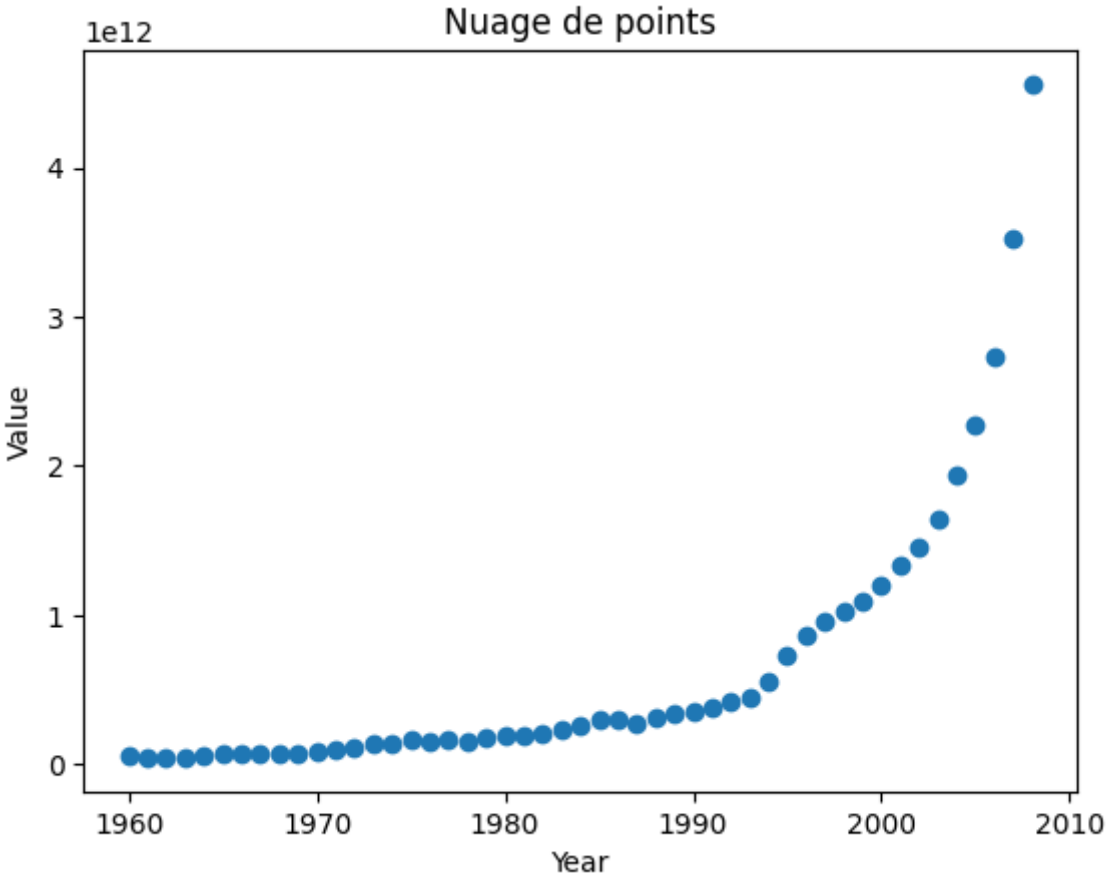
```
<class 'pandas.core.frame.DataFrame'>
Index: 49 entries, 0 to 48
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Year    49 non-null     int64
 1   Value   49 non-null     float64
dtypes: float64(1), int64(1)
memory usage: 1.1 KB
```

- Statistiques descriptives

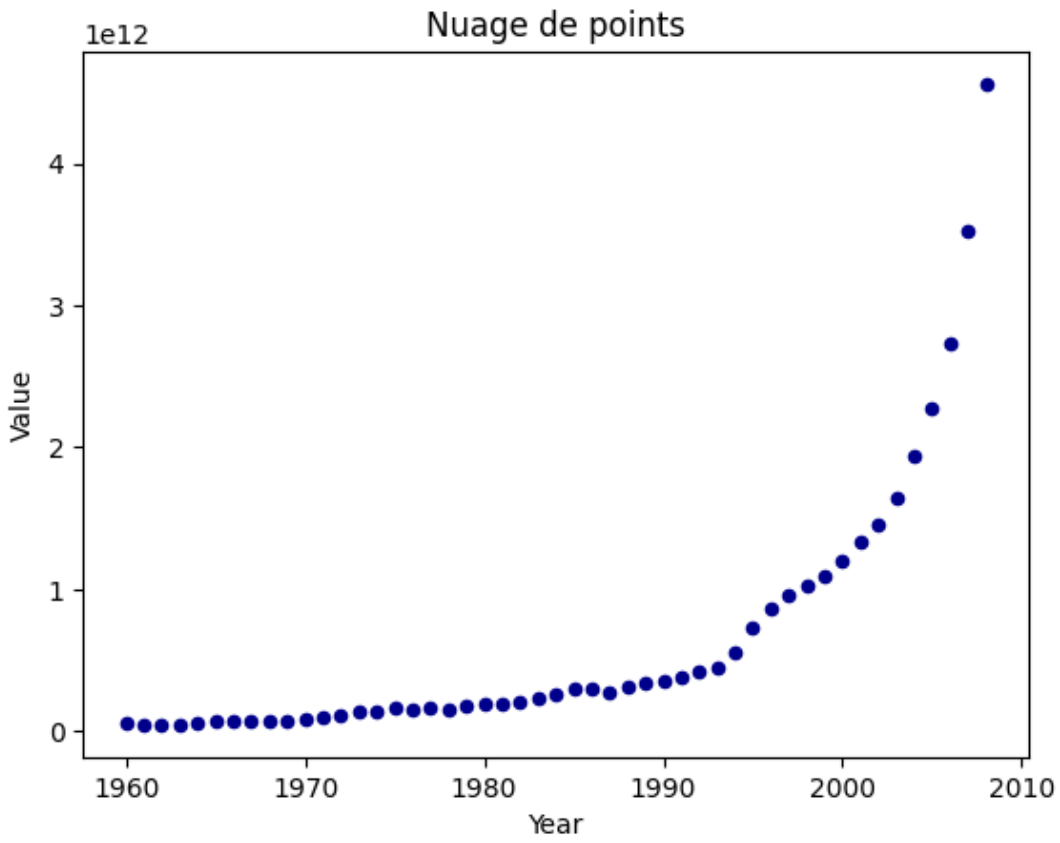
[83]:

	Year	Value
count	49.00000	4.900000e+01
mean	1984.00000	6.558925e+11
std	14.28869	9.455853e+11
min	1960.00000	4.668518e+10
25%	1972.00000	1.121598e+11
50%	1984.00000	2.580821e+11
75%	1996.00000	8.608441e+11
max	2008.00000	4.558431e+12

- Afficher la nuage des points matplotlib



- Afficher la nuage des points Pandas



3. Entraînement de modèle

- Division data en 2 data sets : traitement *08% et test 20%

```
[89]: X = data[['Year']].values
      Y = data['Value'].values
      scaler = StandardScaler()
      X = scaler.fit_transform(X)
      x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
      print(x_train.shape)
      print(x_test.shape)
      print(y_train.shape)
      print(y_test.shape)
```

```
(39, 1)
(10, 1)
(39,)
(10,)
```

- Entraîner le modèle
  - La régression linière

```
# la régression linière
regressor_linière = LinearRegression()
regressor_linière.fit(x_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

- La régression linière polynomiale

```
#La régression linière polynomiale
degree = 5
poly_features = PolynomialFeatures(degree=degree)
x_poly_train = poly_features.fit_transform(x_train)
x_poly_test = poly_features.transform(x_test)
poly_model = LinearRegression()
poly_model.fit(x_poly_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

4. Prédiction sur les données de test
   - La régression linière

```
# La régression linière
y_test_pred_lin = regressor_linière.predict(x_test)
y_test_pred_lin
```

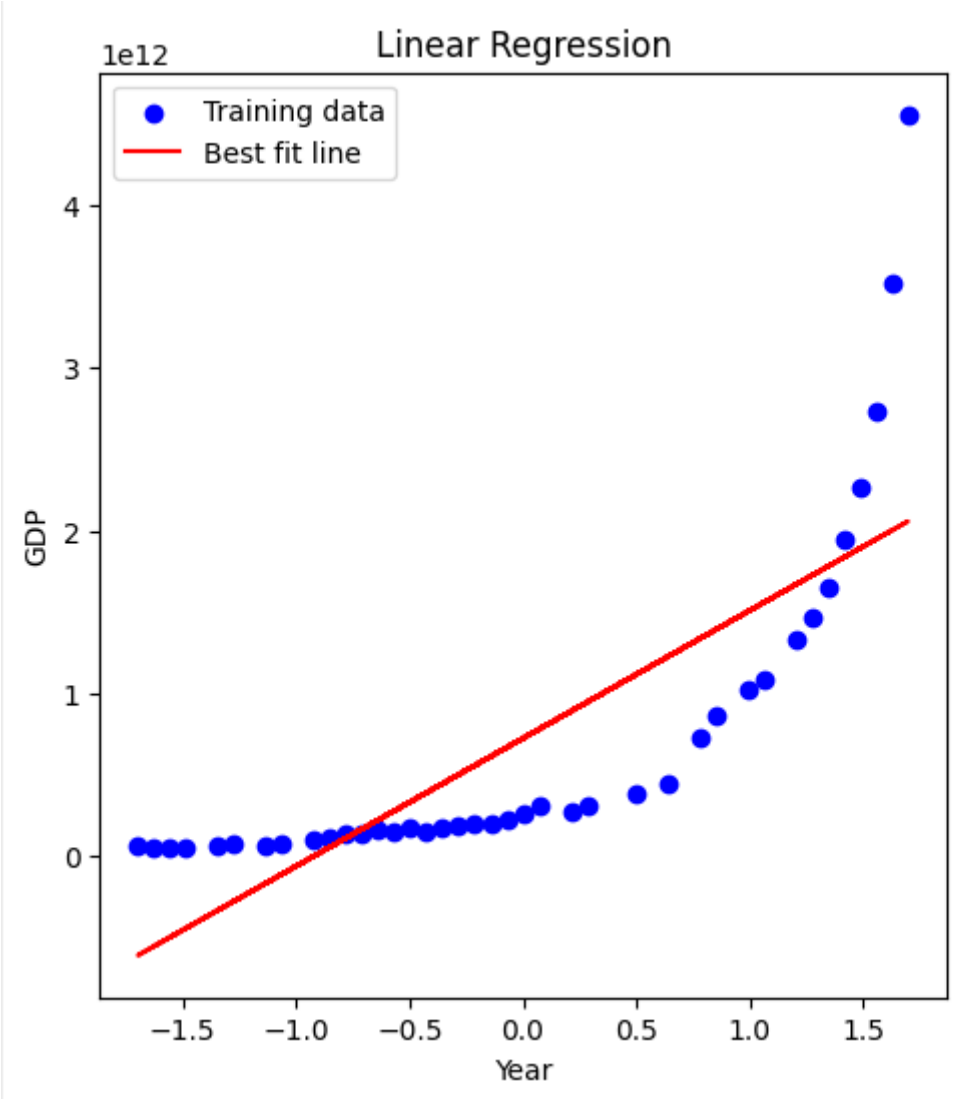
```
array([ 1.00252962e+12, -3.86796410e+11,  8.35810495e+11,  1.05810266e+12,
        1.16924874e+12,  1.44711395e+12,  1.28039482e+12,  1.61383307e+12,
       -2.20077286e+11, -5.33581628e+10])
```

   - La régression linière polynomiale

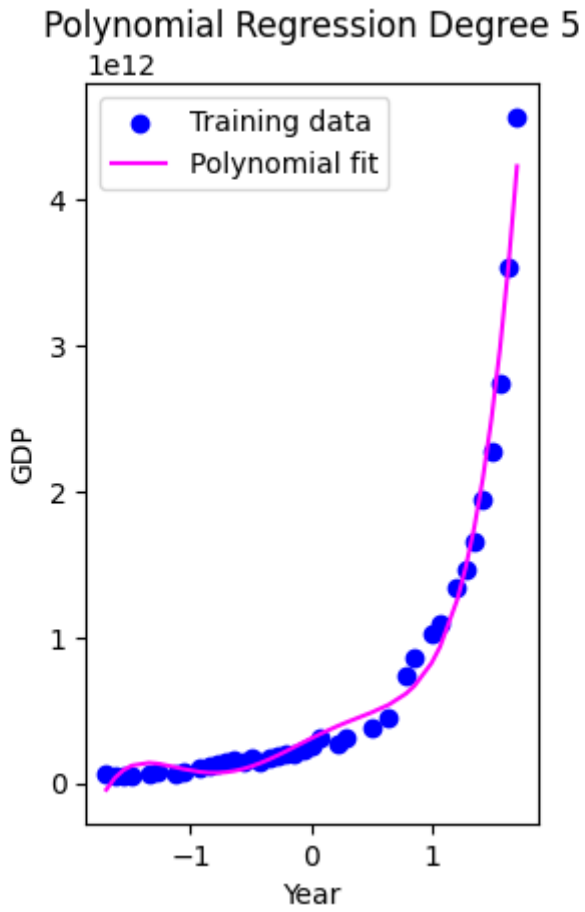
```
#La régression linière polynomiale
y_train_pred_poly = poly_model.predict(x_poly_train)
y_test_pred_poly = poly_model.predict(x_poly_test)
y_test_pred_poly
```

```
array([4.37091803e+11, 1.35396593e+11, 3.63401570e+11, 4.59810171e+11,
       5.07605506e+11, 7.32534316e+11, 5.69324125e+11, 1.07434981e+12,
       1.22572011e+11, 8.48038686e+10])
```

5. Visualiser le résultat de la régression sous forme d'un graphe
   - La régression linière



- La régression linière polynomiale



6. Évaluation de modèle

La régression linière

```
La performance du modèle sur la base dapprentissage
Mean Squared Error (MSE): 2.9890893352317472e+23
Root Mean Squared Error (RMSE): 546725647398.37726
Mean Absolute Error (MAE): 513553890488.1462
```

La régression linière polynomiale

```
La performance du modèle sur la base dapprentissage
Mean Squared Error (MSE): 1.0599858156118995e+22
Root Mean Squared Error (RMSE): 102955612552.7841
Mean Absolute Error (MAE): 83638911270.11707
```