Rapport de Projet

Premier Modèle IA (Régression linéaire simple, multiple et polynomiale avec la méthode normale et la méthode Scikitlearn).

27/01/22

Sommaire :

1 – Introduction au Machine Learning :

En machine Learning il existe plusieurs types d'apprentissages :

- L'apprentissage **supervisé** (Supervised Learning)
- L'apprentissage **non supervisé** (Unsupervised Learning)
- L'apprentissage par **renforcement** (Reinforcement Learning)

Ici pour notre Premier Modèle IA(régression linéaire simple , multiple , et polynomiale avec et sans la bibliothèque Scikit-learn) nous utiliserons l'apprentissage supervisé.

L'idée centrale du Machine Learning, c'est de laisser la machine trouver quels sont les paramètres de notre modèle qui minimisent la Fonction Coût.

Pour cela nous commençons par extraire une ou des features (X) de notre DataSet ainsi qu'identifier ce qui sera notre target (Y) :

Une fois cela fait on peut commencer par la première méthode : Régression linéaire simple :

1. Le Dataset :

En Machine Learning, tout démarre d'un Dataset qui contient nos données. Dans l'apprentissage supervisé, le Dataset contient les questions (x) et les réponses (y) au problème que la machine doit résoudre.

2. Le modèle et ses paramètres :

A partir de ce Dataset, on crée un modèle, qui n'est autre qu'une fonction mathématique. Les coefficients de cette fonction sont les paramètres du modèle.

3. La Fonction Coût :

Lorsqu'on teste notre modèle sur le Dataset, celui-ci nous donne des erreurs. L'ensemble de ces erreurs, c'est ce qu'on appelle la Fonction Coût.

4. L'Algorithme d'apprentissage :

L'idée centrale du Machine Learning, c'est de laisser la machine trouver quels sont les paramètres de notre modèle qui minimisent la Fonction Coût.

2 – Régression Linéaire avec la Méthode Normale :

```
Entrée [1]: import pandas as pd
            from matplotlib import pyplot as plt
            import numpy as np
```

On commence par importer les bibliothèques python nécessaire pour faire la régression

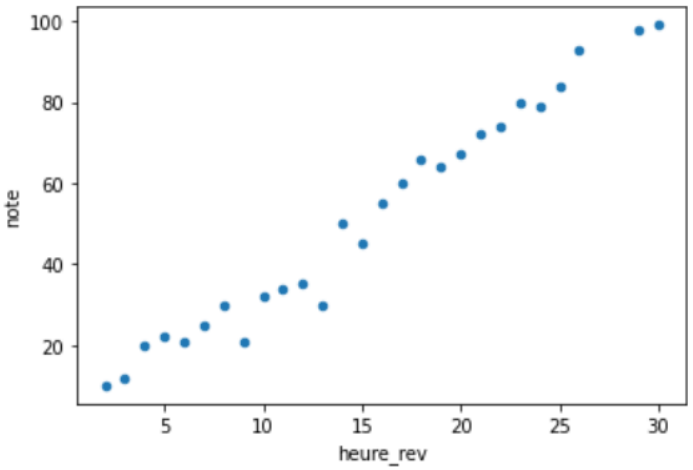
```
x=df[['heure_rev']]
y=np.array(df['note'])

y=y.reshape(y.shape[0],1)

x.shape
```

```
(27, 1)
```

On sélectionne notre Feature (x) et notre Target (y) tout en rajoutant une colonne remplit de 1 à y pour pouvoir que les matrices x et y aient la même taille.

```
df.plot.scatter('heure_rev', 'note')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x2895fbcbca0>
```



On affiche x (heure_rev) et y (note) a l'aide de *plot.Scatter*

```
Entrée [10]: # Déterminer les corrélations les plus fortes
             df.corrwith(df['heure_rev'], method = 'pearson').sort_values(ascending = False)

 Out[10]: heure_rev    1.00000
          note         0.98657
          dtype: float64
```

On vérifie la corrélation la plus forte entre x et y pour être certain de choisir la bonne Feature.

```
Entrée [6]: #matrice X
            X=np.hstack((x,np.ones(x.shape)))
            X[:5]

 Out[6]: array([[2., 1.],
                [3., 1.],
                [4., 1.],
                [5., 1.],
                [6., 1.]])
```

On créer une variable grand X qui contient les valeurs de x ainsi qu'une colonne de 1 de la même taille que nombre de valeurs de x , le tout dans un format array donc une matrice .

Theta

```
Entrée [7]: #initialisation de theta
            theta=np.random.randn(2,1)
            theta.shape

 Out[7]: (2, 1)
```

On initialise Thêta avec des variables aléatoires avec la fonction *randn()* dont les paramètres (2,1), représentent la taille de la matrice qui contiendra les coefficients A et B de la fonction f(x)= Ax+B.

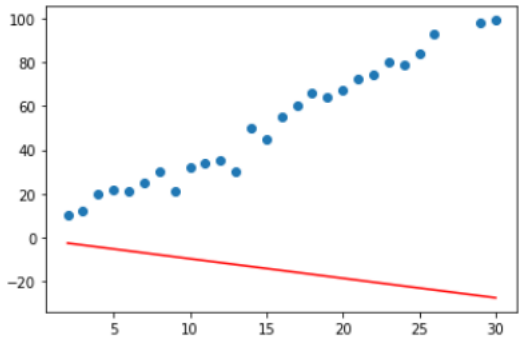
Modèle ¶

```
Entrée [8]: def model(X,theta):
                return X.dot(theta)
            model(X,theta)

 Out[8]: array([[ -2.47280256],
                [ -3.36139766],
                [ -4.24999275],
                [ -5.13858785],
                [ -6.02718295],
                [ -6.91577805],
                [ -7.80437315],
                [ -8.69296825],
                [ -9.58156335],
                [-10.47015845],
                [-11.35875354],
                [-12.24734864],
                [-13.13594374],
                [-14.02453884],
                [-14.91313394],
                [-15.80172904],
                [-16.69032414],
                [-17.57891923],
                [-18.46751433],
                [-19.35610943],
                [-20.24470453],
                [-21.13329963],
                [-22.02189473],
                [-22.91048983],
                [-23.79908493],
                [-26.46487022],
                [-27.35346532]])
```

On définit le model avec la fonction *Def model*(X,Theta) qui retourne le produit matriciel de X par Theta grâce a la fonction *dot.*

```
Entrée [9]: plt.scatter(x,y)
            plt.plot(x,model(X,theta), color='r')
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x2c10f8cee80>]
```



On affiche une nouvelle fois x et y mais cette fois ci avec notre model X.

Fonction coût ¶

```
Entrée [10]: #Fonction Cout

             def cost_fonction(X,y,theta):
                 m=len(y)
                 return 1/(2*m)*np.sum((model(X,theta)-y)**2)
```

```
Entrée [11]: cost_fonction(X,y,theta)
```

```
Out[11]: 2713.1088959024696
```

On créer la fonction Coût.

Gradient et descente de gradient

```
Entrée [12]: def grad (X,y,theta):
                 m=len(y)
                 return 1/m * X.T.dot(model(X,theta)-y)
```

```
Entrée [13]: def gradient_descent(X,y,theta,learning_rate,n_iterations):
                 cost_history=np.zeros(n_iterations)
                 for i in range(0,n_iterations):
                     theta=theta-learning_rate* grad(X,y,theta)
                     cost_history[i]=cost_fonction(X,y,theta)
                 return theta,cost_history
```

On créer notre fonction Gradient ainsi que notre fonction Descente de gradient.

Theta Final

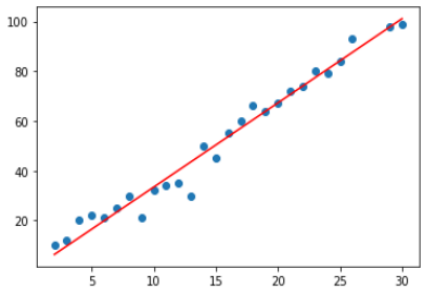
Entrée [14]:
```
n_iterations= 30
learning_rate =0.001
theta_final,cost_history = gradient_descent(X,y,theta,learning_rate, n_iterations)
```

Entrée [15]:
```
theta_final
```

Out[15]:
```
array([[ 3.38701076],
       [-0.46927132]])
```

Entrée [16]:
```
predictions=model(X,theta_final)
plt.scatter(x,y)
plt.plot(x,predictions,c='r')
```

Out[16]: `[<matplotlib.lines.Line2D at 0x2c10f983d90>]`



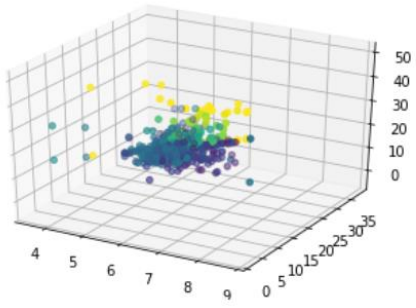
On définit notre Thêta « final » c'est-à-dire notre Thêta après avoir effectué une descente de gradient qui nous donne les meilleur coefficients A et B.

On affiche ensuite le Model.

Visualisation 3D

Entrée [18]:
```
ax = plt.axes(projection='3d')
ax.scatter(x1, x2, y, c=y)
ax.scatter(x1, x2, predictions, c=y)
```

Out[18]: `<mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x1ff37e43850>`

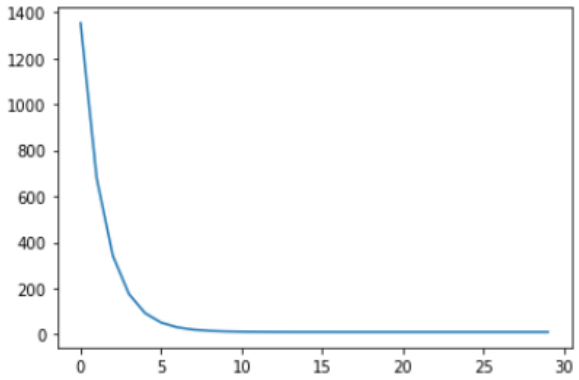


On peut par exemple afficher un graphique 3D avec *Scatter* qui prend comme paramètres x1 x2 et Y afin de visualiser nos données.

Courbes d'apprentissage

```
Entrée [17]: plt.plot(range(n_iterations),cost_history)
```

```
Out[17]: [<matplotlib.lines.Line2D at 0x2c10f9dfa90>]
```



On affiche la courbe d'apprentissage du modèle.

```
Entrée [18]: def coef_determination(y,pred):
                 u=((y-pred)**2).sum()
                 v=((y-y.mean())**2).sum()
                 return 1-u/v
```

```
Entrée [19]: coef_determination(y,predictions)
```

```
Out[19]: 0.9730795470485297
```

Enfin on calcul le coefficient de détermination pour évaluer l'efficacité de notre modèle.

3 – Régression Multiple avec la Méthode Normale

On peut reprendre la méthode de régression linéaire simple pour effectuer notre régression multiple en modifiant des paramètres :

```
Entrée [5]: x1=df[['RM']]
            x2=df[['LSTAT']]
            y=np.array(df[['MEDV']])

            y=y.reshape(y.shape[0],1)
            y.shape

  Out[5]: (506, 1)
```

```
Entrée [6]: #matrice X
            X=np.hstack((x1,x2,np.ones(x1.shape)))
            X.shape

  Out[6]: (506, 3)
```

On choisit cette fois ci plusieurs Target (x), dans ce cas x1 et x2.

Theta

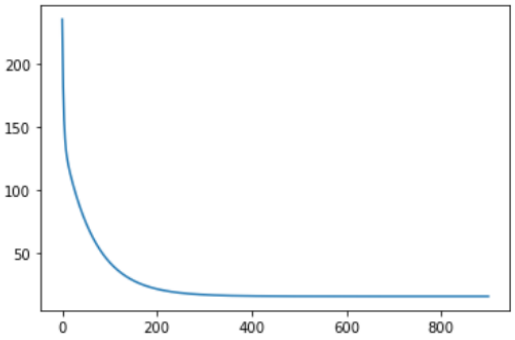
```
Entrée [7]: #initialisation de theta
            theta=np.random.randn(3,1)
            theta

  Out[7]: array([[-0.04375142],
                 [ 0.09137191],
                 [ 0.1216061 ]])
```

On modifie les paramètres de Thêta : (3,1)

Courbe d'apprentissage ¶

```
Entrée [19]: plt.plot(range(n_iterations),cost_history)
```

```
Out[19]: [<matplotlib.lines.Line2D at 0x1ff3835d430>]
```




```
Entrée [20]: coef_determination(y,predictions)
```

```
Out[20]: 0.6382335062686587
```

Enfin on affiche la courbe d'apprentissage et le Coefficient de détermination.

4 -Régression Polynomiale avec la Méthode normale :

On peut reprendre la méthode de régression multiple pour effectuer notre régression polynomiale en modifiant des paramètres :

```
Entrée [4]: # Déterminer les corrélations les plus fortes
            df.corrwith(df['Salary'], method = 'pearson').sort_values(ascending = False)
```

```
Out[4]: Salary    1.000000
        Level     0.817949
        dtype: float64
```

```
Entrée [5]: x=df[['Level']]
            y=np.array(df[['Salary']])
            y.shape
```

```
Out[5]: (10, 1)
```

```
Entrée [6]: #matrice X
            X=np.hstack((x**2,x,np.ones(x.shape)))
            X.shape
```

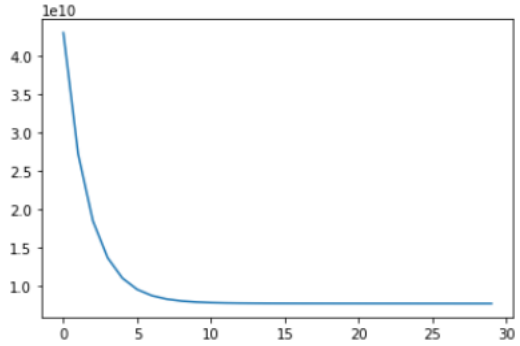
```
Out[6]: (10, 3)
```

Après avoir définit notre Feature (x) et notre Target(y) on créer notre variable X qui contient nos valeur de x au carré cette fois , en plus d'une colonne de 1 de la même taille que nombre de valeurs de x , le tout dans un format Array donc une matrice .

Courbe d'apprentissage

```
Entrée [17]: plt.plot(range(n_iterations),cost_history)
```

```
Out[17]: [<matplotlib.lines.Line2D at 0x1dcd426e3a0>]
```




```
Entrée [18]: def coef_determination(y,pred):
                 u=((y-pred)**2).sum()
                 v=((y-y.mean())**2).sum()
                 return 1-u/v
```

```
Entrée [19]: coef_determination(y,predictions)
```

```
Out[19]: 0.8074315569362831
```

On affiche finalement notre courbe d'apprentissage ainsi que notre coefficient de détermination.

5 – Régression Polynomiale « Qualité vin rouge »

Theta Final

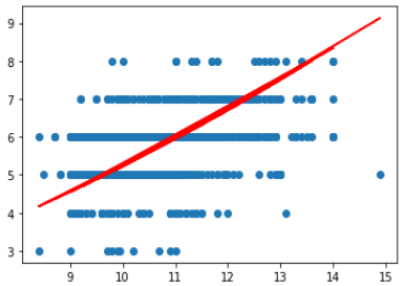
```
Entrée [14]: 
n_iterations= 300
learning_rate =0.00001
theta_final,cost_history = gradient_descent(X,y,theta,learning_rate, n_iterations)
```

```
Entrée [15]: theta_final
```

```
Out[15]: array([[0.02043089],
                [0.28778999],
                [0.30970857]])
```

```
Entrée [16]: predictions=model(X,theta_final)
plt.scatter(x,y)
plt.plot(x,predictions,c='r')
```

```
Out[16]: [<matplotlib.lines.Line2D at 0x1ce781719d0>]
```



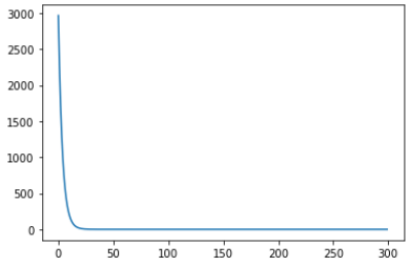
On remarque que le jeu de données n’est pas adapté pour une régression de type Polynomiale.

Courbe d'apprentissage ¶

La regression ne fonctionne pas car les données sont qualitatives et non quantitatives , il serait pertinent de faire une classification ou une regression logistique.

Entrée [20]:
```
plt.plot(range(n_iterations),cost_history)
```

Out[20]: [<matplotlib.lines.Line2D at 0x1ce781d4580>]



Entrée [17]:
```
def coef_determination(y,pred):
    u=((y-pred)**2).sum()
    v=((y-y.mean())**2).sum()
    return 1-u/v
```

Entrée [18]:
```
coef_determination(y,predictions)
```

Out[18]: -0.02810745146807081

Le coefficient est même négatif, le type de régression n’est pas adapté il serait plus judicieux d’appliquer une méthode de classification ou une régression logistique.

Régressions avec Sklearn

L'objectif est de refaire les régressions précédentes avec la bibliothèque Sklearn.

Régression linéaire avec Sklearn

1- Importation des bibliothèques et des modules nécessaires

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

2- – On importe les données du fichier csv que l'on place dans la variable « df ». On afficher les 5 premières valeurs avec .head() et les 5 dernières avec .tail() .

```
df = pd.read_csv('Data/reg_simple.csv')
df.head()
```

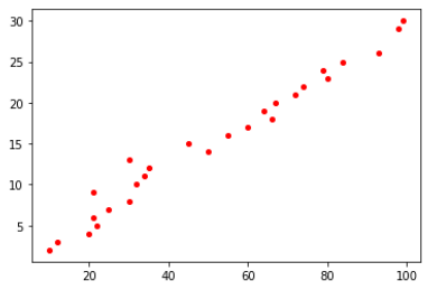
	heure_rev	note
0	2	10
1	3	12
2	4	20
3	5	22
4	6	21

```
df.tail()
```

	heure_rev	note
22	24	79
23	25	84
24	26	93
25	29	98
26	30	99

3- On peut visualiser données avec matplotlib :

```
plt.plot(df['note'], df['heure_rev'], 'ro', markersize=4)
plt.show()
```



4- On fractionne notre jeu de données en deux : la variable X représente heure_rev et la variable Y les notes :

```
X = df.iloc[:, :-1].values
y = df.iloc[:, 1].values
```

5- On fractionne ensuite le jeu de données en variables X_train, X_test et y_train et y_test en prenant 80% pour le train et 20% pour le test. Le test servira à valider (ou non) notre phase d'entraînement :

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

6- On lance le modèle de régression linéaire avec la fonction LinearRegression() :

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

7- On peut observer les résultats de prédiction de la variable Test. On crée une variable y_pred où l'on invoque la fonction .predict sur la variable regressor avec les données de X_test :

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

8- On peut visualiser notre régression :

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

Régression multiple avec Sklearn

On applique les mêmes étapes que pour la régression linéaire, sauf que notre X contient plus de données.

Une fois les variables X et y définies, on procède à la fragmentation des données en set de train et de test.

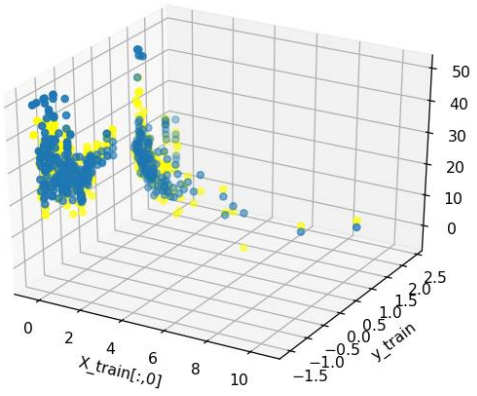
On peut alors standardiser nos valeurs :

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

L'entraînement des modèles est ensuite identique à celui de la régression précédente.
On peut alors visualiser en 3D nos résultats :

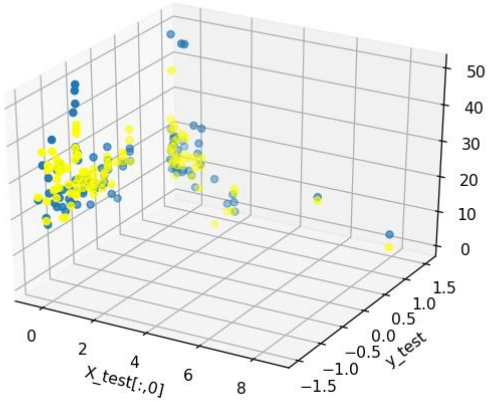
Avec les données train :

```
from mpl_toolkits.mplot3d import Axes3D
%matplotlib notebook

fig= plt.figure()
ax=fig.add_subplot(111,projection='3d')
plt.xlabel("X_train[:,0]")
plt.ylabel("y_train")
ax.scatter(X_train[:,0],X_train[:,2],y_train)
ax.scatter(X_train[:,0],X_train[:,2],y_pred_train,color = 'yellow')
```



Avec les données test :

```
from mpl_toolkits.mplot3d import Axes3D
%matplotlib notebook

fig= plt.figure()
ax=fig.add_subplot(111,projection='3d')
plt.xlabel("X_test[:,0]")
plt.ylabel("y_test")
ax.scatter(X_test[:,0],X_test[:,2],y_test)
ax.scatter(X_test[:,0],X_test[:,2],y_pred_test,color = 'yellow')
```



On peut alors évaluer la qualité de notre modèle :

```
# On utilise RMSE et R²-score.
from sklearn.metrics import r2_score

# Evaluation du modèle (training set)

y_train_predict = regressor.predict(X_train)
rmse = (np.sqrt(mean_squared_error(y_train, y_train_predict)))
r2 = r2_score(y_train, y_train_predict)

print("La performance du modèle pour training set")
print("--------------------------------------")
print('RMSE est de : {}'.format(rmse))
print('R2 score est de : {}'.format(r2))
print("\n")

# Evaluation du modèle (testing set)

y_test_predict = regressor.predict(X_test)
# root mean square error of the model
rmse = (np.sqrt(mean_squared_error(y_test, y_test_predict)))

# Score r-squared pour le modèle
r2 = r2_score(y_test, y_test_predict)

print("La performance du modèle pour testing set")
print("--------------------------------------")
print('RMSE est de : {}'.format(rmse))
print('R2 score est de : {}'.format(r2))
```

Ce qui nous renvoie :

```
La performance du modèle pour training set
--------------------------------------
RMSE est de : 4.396188144698283
R2 score est de : 0.7730135569264233


La performance du modèle pour testing set
--------------------------------------
RMSE est de : 5.783509315085131
R2 score est de : 0.5892223849182514
```

Enfin, la fonction *mean_squared_error* de sklearn nous renvoie le RMSE :

```python
# Root Mean Square Error

from sklearn.metrics import mean_squared_error
import math

MSE = mean_squared_error(y_test, y_test_predict)

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:\n")
print(RMSE)
```

```
Root Mean Square Error:

5.783509315085131
```

Régression polynomiale avec Sklearn

On importe les bibliothèques et les modules nécessaires :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
```

On enregistre la data dans un variable appelée « df » dont on peut afficher les premiers éléments :

```
df = pd.read_csv("Data/Position_Salaries.csv")

df.head()
```

	Position	Level	Salary
0	Project Analyste	1	45000
1	Ingenieur	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000

On partage les données en X(Level) et y (Salary) :

```
x = df.iloc[:, 1:-1].values
y = df.iloc[:, -1].values
```

Ensuite il faut fractionner nos variables en X_train et X_test et y_train et y_test. On utilise pour cela avec la fonction *train_test_split* de sklearn et l'on sélectionne 80% pour le train et 20% pour le test :

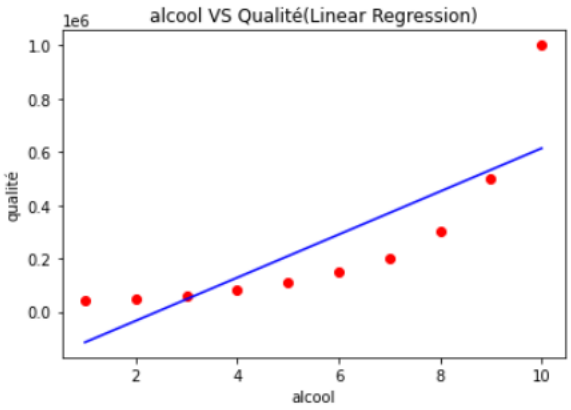
```
#fractionner jeu de données

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2)
```

On lance alors le modèle de régression linéaire avec *LinearRegression* :

```
from sklearn.linear_model import LinearRegression
Reg = LinearRegression()
Reg.fit(x, y)
```

Visualisons nos données :

```
plt.scatter(x, y, color = 'red')
plt.plot(x, Reg.predict(x), color = 'blue')
plt.title('alcool VS Qualité(Linear Regression)')
plt.xlabel('alcool')
plt.ylabel('qualité')
plt.show
```



On lance le modèle de régression polynomiale :

```
poly_reg = PolynomialFeatures(degree = 3)
X_poly_train =poly_reg.fit_transform(X_train)
X_poly_test =poly_reg.fit_transform(X_test)
```

```
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly_train, y_train)
```

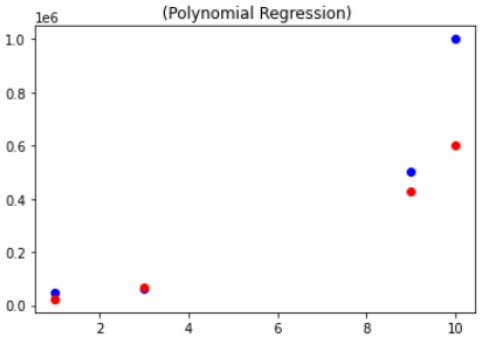
```
LinearRegression()
```

On procède à la phase de prédiction :

```
y_pred = lin_reg_2.predict(X_poly_test)
```

En visualisant notre jeu de données, nous obtenons le graphique suivant (du fait de la faible quantité du jeu de données) :

```
plt.scatter(X_test, y_test, color = 'blue')
plt.scatter(X_test, y_pred, color = 'red')
plt.title(' (Polynomial Regression)')
plt.xlabel('')
plt.ylabel('')
plt.show()
```



Evaluons notre modèle avec le coefficient de détermination avec *mean squarred error* avec la fonction *r2_score* de *sklearn.metrics* :

```
from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print('R2 score est de : {}'.format(r2))
```

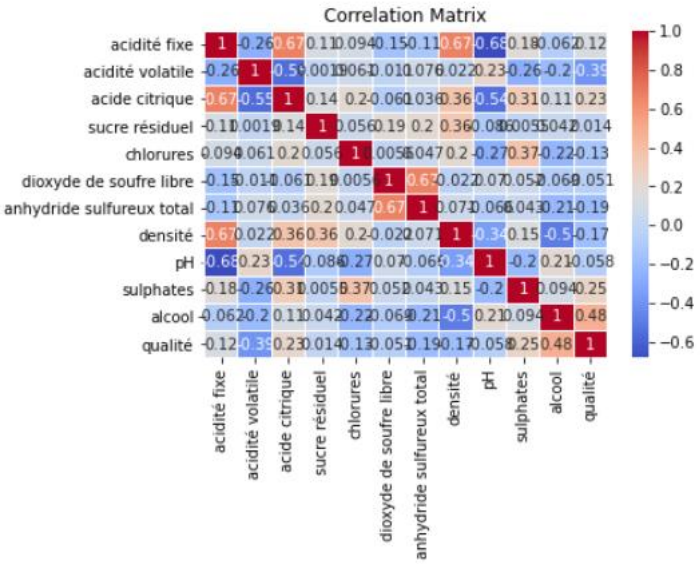
```
R2 score est de : 0.8688814279352782
```

Régression polynomiale avec les données "qualite_vin_rouge.csv"

Cette régression est dans l'ensemble identique à la précédente, aussi nous ne préciserons que les légères variations.

```
# matrice de corrélation

plt.title("Correlation Matrix")
sns.heatmap(data=df.corr(),annot=True,cmap='coolwarm',linewidths=0.1)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b34ca4e6d0>
```

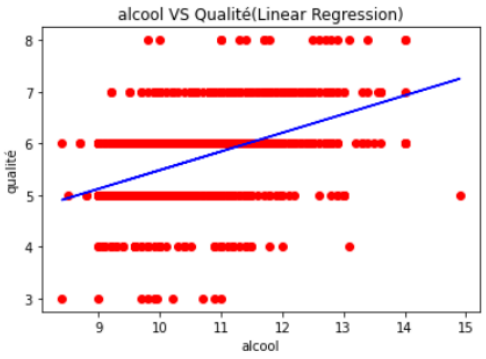


En voyant cette image, on peut sélectionner uniquement la variable "alcool" (0,48).

En faisant une matrice de corrélation nous pouvons constater que la variable la plus intéressante est « alcool », corrélée à 0,48. C'est donc cette variable que nous mettrons dans notre X.

```
#On peut visualiser les données

plt.scatter(x, y, color = 'red')
plt.plot(x, LinReg.predict(x), color = 'blue')
plt.title('alcool VS Qualité(Linear Regression)')
plt.xlabel('alcool')
plt.ylabel('qualité')
plt.show
```

```
<function matplotlib.pyplot.show(*args, **kw)>
```



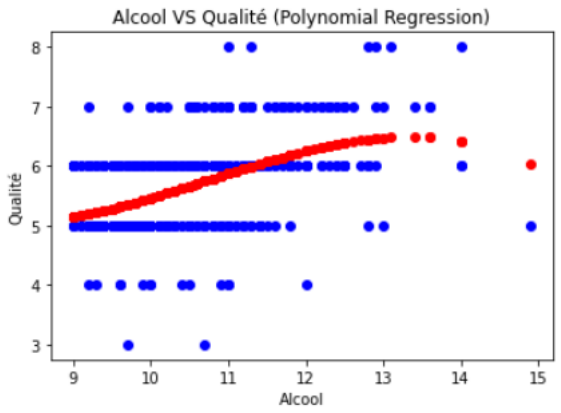
```
# On lance le modèle de régression polynomiale

poly_reg = PolynomialFeatures(degree = 3)
X_poly_train =poly_reg.fit_transform(X_train)
X_poly_test =poly_reg.fit_transform(X_test)
```

```
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly_train, y_train)
```

```
LinearRegression()
```

```
# Phase de prédiction
y_pred = lin_reg_2.predict(X_poly_test)
```

```
plt.scatter(X_test, y_test, color = 'blue')
plt.scatter(X_test, y_pred, color = 'red')
plt.title('Alcool VS Qualité (Polynomial Regression)')
plt.xlabel('Alcool')
plt.ylabel('Qualité')
plt.show()
```



Nous pouvons évaluer notre modèle avec avec le coefficient de détermination avec *mean squarred error* avec la fonction *r2_score* de *sklearn.metrics* :

```
# Evaluation du modèle avec coef de détermination avec mean squarred error

from sklearn.metrics import r2_score
r2 = r2_score(y_test, y_pred)
print('R2 score est de : {}'.format(r2))
```

```
R2 score est de : 0.18766066772337553
```

Conclusion :

Les résultats semblent plus précis avec la méthode sklearn. Cependant, la méthode normale permet de garder une certaine maîtrise sur le jeu de données et une meilleur souplesse . Ce qui nous incite à dire qu'en règle générale la bibliothèque sklearn et ses modules sont suffisants, mais qu'il faudra privilégier la méthode normale pour les cas particuliers et précis.

J'ai personnellement appris plein de nouvelles notions avec ce projet qui ne ma d'ailleurs pas semblé facile et j'ai du faire faire a plusieurs difficultés notamment la manipulation des données (iloc,array) ainsi que la création des foncions pour la méthode normale. Je dois revoir et m'entrainer avec la méthode Scikit-learn pour être plus à l'aise.