Régression linéaire simple :

Le but de la régression simple est d'expliquer une variable Y à l'aide d'une variable X.
« Y en fonction X »

8 étapes de réalisation :

1.Tout d'abord on commence par importer la bibliothèque nécessaire :

```
import numpy as np #destine a manipuler des matrices ou tableaux
import matplotlib.pyplot as plt #tracer et visualiser des donnees
import pandas as pd #manipulation et analyses de donnees
from sklearn.metrics import mean_squared_error, mean_absolute_error #machine learning
```

2.Puis on importe le jeu de données :

```
dataset = pd.read_csv('Salaire_Experience.csv') #ici 'Salaire_Experience.csv' est le nom du fichier a traiter
dataset.head() #visualisation
```

3.Il nous faut maintenant définir X et Y dans ce jeu de données :

```
#avec la fontion .iloc[]
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 1].values

#avec la fonction df[]
x = df['note']
y = df['heure_rev']
```

avec la fonction iloc on définis par rapport à un numéro de colonne ou de ligne

ici on extrait directement les valeurs de la colonne 'note' de la dataframe 'df'
Attention cette méthode peut générer des erreurs

4.Maintenant que les bases sont définis on fractionne ces données en deux groupes :
entraînements et test :

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

ici test_size nous permet de définir le groupe de test à 20% (0,2/1) le groupe d'entraînement prends automatiquement ce qu'il reste

5.On crée le modèle de régression linéaire et on l'utilise sur les données d'entraînement que l'on à créée dans l'étape précédente :

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

on l'a appelé ici 'regressor'

6.Il est désormais possible de prédire les résultat du modèle sur l'ensemble du test :

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

ici on prédit Y par rapport a X

7.Visualisons nos résultats :

d'entraînement :

```
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()
```

de test :

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

Même chose mais ici on utilise les données de test dans plt.scatter() :
X_train et Y_train

8.Enfin, évaluons notre modèle :

```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)

print(mse)
print(mae)
```

On utilise ici les fonctions : mean_squared_error() et mean_absolute_error()

Régression linéaire multiple :

Le but de la régression multiple est d'expliquer une variable Y à l'aide de plusieurs variables X.
« Y en fonction nX »

Important :

La fonction .iloc

```
dataset.iloc[0:5, 0:9].values
```

Nom de la dataframe
De la LIGNE 0 à 5 sans inclure la 5
De la COLONNE 0 à 9 sans inclure la 9

Il est donc important de savoir à quoi ressemble notre dataframe

3.pour réaliser une fonctionne linéaire multiple on va reprendre à partir de l'étape 3 de la fonction linéaire simple en sélectionnant plusieurs colonnes pour la valeur X :

```
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
print(X.shape)
```

« x équivaut à toutes les lignes de toutes les colonnes sauf la dernière »
« y équivaut à toutes les lignes des colonnes 0 à 4 en excluant la 4 »

4.On va désormais transformer les valeurs qualitatives en valeurs numérique :

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
labelencoder = LabelEncoder()

X[:, 3] = labelencoder.fit_transform(X[:, 3])
print(X[:, 3])
```

il est important ici d'écraser les valeurs de la colonne 3 pour ne garder que les valeurs (0,1,2,...) qu'on leurs à donné

5. Fractionnement des 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, random_state = 0)
```

cette étape est la même que pour la régression linéaire simple (étape 4)

6.On va désormais standardiser les variables d'entraînement et de test :

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

cela permet de réduire leurs impact en cas de trop gros écarts de valeurs

7.création du modèle de régression linéaire et entraînement sur les données prévu pour ça :

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

cette étape est la même que pour la régression linéaire simple (étape 5)

8.Prédiction des résultats du modèle sur l'ensemble du test :

```
y_pred_train = regressor.predict(X_train)
y_pred_test = regressor.predict(X_test)
```

ici on prédit les deux dans la même étape

9.Visualisation des résultats :

il sera nécessaire ici de visualiser en 3D comme nous avons plusieurs valeurs

Entraînement :

```
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')
```

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 remarque que c'est similaire à la régression linéaire simple mais ici ça sera de la visualisation en 3D

10.Enfin nous pouvons évaluer notre modèle :

```
mse2 = mean_squared_error(y_test, y_pred_test)
print(mse2)
```

<u>Régression polynomiale :</u>

Ici nous n'aurons pas assez de données pour effectuer une phase de fraction et de test.

Importer les modules et la data puis définir X et Y comme vu au dessus

4.création du modèle de régression linéaire et son entraînement :

```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, y)
```

5.visualisation :

```
plt.scatter(X, y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Linear Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

6. On va créer un modèle de régression polynomiale et essayer avec plusieurs valeurs afin de visualiser la meilleure valeur pour le degré :

```
from sklearn.preprocessing import PolynomialFeatures

colors=['blue','yellow','green','pink','black'] #les couleurs des différentes courbes
plt.scatter(X, y, color = 'red')

plt.title('Polynomial Regression')
plt.xlabel('Position level')
plt.ylabel('Salary')
for i in range(1,6):#boucle pour recommencer l'entrainement 5 fois (range1,6)

    poly_reg = PolynomialFeatures(degree = i)
    X_poly = poly_reg.fit_transform(X)
    poly_reg.fit(X_poly, y)
    lin_reg_2 = LinearRegression()
    lin_reg_2.fit(X_poly, y)

    plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = colors[i-1],label='degre: %s' %i)

plt.legend()
plt.show()
```

plus la courbe épouse la forme des points plus elle est précise

7.Enfin on prédit une valeur quelconque qui n'est pas présente dans la data :

```
valeur=np.array([8]).reshape(-1, 1)

print('Predicting a new result with Polynomial Regression')
print(lin_reg_2.predict(poly_reg.fit_transform(valeur)))
```

si on voulais comparer les résultat avec les résultat d'un valeur de régression linéaire simple ajouter :

```
print('Predicting a new result with Linear Regression')
print(lin_reg.predict(valeur))
```