Travaux Pratiques
Apprentissage Automatique I

TP #1 – Régression

Professeur

Abdelouahab Moussaoui

Partie 1 — Régression en action

Dans cette partie nous allons apprendre à réaliser avec R et Python des modèles de régression linéaire Simple, Multiple et Polynomiale

Exercice 1 — Application de la Régression Simple

En statistique, ***la régression linéaire*** est une approche linéaire de modélisation de la relation entre une réponse scalaire (ou variable dépendante) et une ou plusieurs variables explicatives (ou variables indépendantes). Dans notre exemple, nous allons passer par la régression linéaire simple.
La régression linéaire simple est de la forme $y = w.x + b$, où y est la variable dépendante, x est la variable indépendante, w et y sont les paramètres de formation qui doivent être optimisés pendant le processus de formation pour obtenir des prédictions précises.
Appliquons maintenant l'apprentissage automatique pour former un ensemble de données afin de prédire le ***salaire*** à partir ***des années d'expérience*** .

Application avec R

Exécuter le script suivant dans R.

```
# Simple Linear Regression

# Importing the dataset
dataset = read.csv('Salary_Data.csv')

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')

library(caTools)

# Split dataset into training dataset (ratio 2/3) and testing dataset (ratio 1/3)
set.seed(123)
split = sample.split(dataset$Salary, SplitRatio = 2/3)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
# training_set = scale(training_set)
# test_set = scale(test_set)

# Fitting Simple Linear Regression to the Training set
regressor = lm(formula = Salary ~ YearsExperience, data = training_set)

# Predicting the Test set results
y_pred = predict(regressor, newdata = test_set)

# Visualising the Training set results
library(ggplot2)

ggplot() +
  geom_point(aes(x = training_set$YearsExperience, y = training_set$Salary),
             colour = 'red') +
  geom_line(aes(x = training_set$YearsExperience, y = predict(regressor, newdata =
             training_set)), colour = 'blue') +
  ggtitle('Salary vs Experience (Training set)') +
```

```
  xlab('Years of experience') +
  ylab('Salary')

# Visualising the Test set results
library(ggplot2)
ggplot() +
  geom_point(aes(x = test_set$YearsExperience, y = test_set$Salary),
             colour = 'green') +
  geom_line(aes(x = training_set$YearsExperience, y = predict(regressor, newdata =
             training_set)), colour = 'blue') +
  ggtitle('Salary vs Experience (Test set)') +
  xlab('Years of experience') +
  ylab('Salary')
```

Application avec Python

Exécuter le script suivant dans anaconda python.

Étape 1 : Importation des bibliothèques

Dans cette première étape, nous allons importer la bibliothèque ***pandas*** qui sera utilisée pour stocker les données dans un Pandas DataFrame. Le ***matplotlib*** est utilisé pour tracer des graphiques.

```
import numpy as np
import matplotlib.pyplot as plt
from pandas as pd
```

Étape 2 : Importer le jeu de données

Dans cette étape, nous allons charger l'ensemble de données qui contient les données sous le nom "Salary_Data.csv". La variable ***X*** stockera les « ***années d'expérience*** » et la variable ***Y*** stockera le « ***salaire*** » **.** Le `dataset.head(5)` est utilisé pour visualiser les 5 premières lignes des données.

```
dataset = pd.read_csv('Salary_Data.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].valuesdataset.head(5)

>>
YearsExperience Salary
1.1             39343.0
1.3             46205.0
1.5             37731.0
2.0             43525.0
2.2             39891.0
```

Étape 3 : Diviser l'ensemble de données en ensemble d'apprentissage et en ensemble de test

Dans cette étape, nous devons diviser l'ensemble de données en l'ensemble d'apprentissage, sur lequel le modèle de régression linéaire sera formé et l'ensemble de test, sur lequel le modèle formé sera appliqué pour visualiser les résultats. En cela, le test_size=0.2 indique que ***20 %*** des données seront conservées en tant ***qu'ensemble de test*** et que les ***80 %*** restants seront utilisés pour la formation en tant ***qu'ensemble d'apprentissage*** .

```
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)
```

Étape 4 : Entraîner le modèle de régression linéaire simple sur l'ensemble d'entraînement

Dans la quatrième étape, la classe *LinearRegression* est importée et est affectée à la variable « ***regressor*** » . La *regressor.fit()* fonction est équipée de **X_train** et ***Y_train*** sur lesquels le modèle sera formé.

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

Étape 5 : Prédire les résultats de l'ensemble de tests

Dans cette étape, la regressor.predict() fonction est utilisée pour prédire les valeurs de l'ensemble de test et les valeurs sont stockées dans la variable y_pred.

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

Étape 6 : Comparaison de l'ensemble de test avec les valeurs prédites

Dans cette étape, un Pandas DataFrame est créé pour comparer les valeurs de salaire de l'ensemble de test d'origine (***y_test***) et des résultats prédits (***y_pred***).

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
>>
Real Values     Predicted Values
109431.0        107621.917107
81363.0         81508.217112
93940.0         82440.849255
55794.0         63788.206401
66029.0         74047.159970
91738.0         89901.906396
```

Nous pouvons voir que les salaires prédits sont très proches des valeurs salariales réelles et on peut conclure que le modèle a été bien formé.

Étape 7 : Visualiser les résultats

Dans cette dernière étape, nous visualisons les résultats des valeurs de salaire ***réel*** et ***prédit*** ainsi que la ligne de régression linéaire sur un graphique tracé.

```
plt.scatter(X_test, y_test, color = 'red')
plt.scatter(X_test, y_pred, color = 'green')
plt.plot(X_train, regressor.predict(X_train), color = 'black')
plt.title('Salary vs Experience (Result)')
plt.xlabel('YearsExperience')
plt.ylabel('Salary')
plt.show()
```

Salaire vs expérience

Dans ce graphique, les valeurs réelles sont tracées en couleur « ***Rouge*** » et les valeurs prédites sont tracées en couleur « ***Vert*** ». La ligne de régression linéaire qui est générée est dessinée en couleur « ***noire*** ».

Exercice 2 — Application de la Régression Multiple

Dans ces données, nous avons les quatre variables indépendantes, à savoir *les dépenses de R&D, l'administration, les dépenses de marketing* et *l'état* . Il y a une variable indépendante, c'est-à-dire *Profit* . Notre travail consiste donc à former le modèle ML avec ces données pour comprendre la corrélation entre chacune des quatre caractéristiques (ou variables indépendantes) et prédire un profit pour une autre nouvelle entreprise avec toutes ces données.

Application avec R

Exécuter le script suivant dans R.

```
# Multiple Linear Regression

# Importing the dataset
dataset = read.csv('50_Startups.csv')

# Encoding categorical data
dataset$State = factor(dataset$State,
                       levels = c('New York', 'California', 'Florida'),
                       labels = c(1, 2, 3))

# Splitting the dataset into the Training set and Test set
# install.packages('caTools')

library(caTools)
set.seed(123)
```

```r
# Split dataset into training dataset (ratio 80%) and testing dataset (ratio 20%)
split = sample.split(dataset$Profit, SplitRatio = 0.8)
training_set = subset(dataset, split == TRUE)
test_set = subset(dataset, split == FALSE)

# Feature Scaling
# training_set = scale(training_set)
# test_set = scale(test_set)

# Fitting Multiple Linear Regression to the Training set
regressor = lm(formula = Profit ~ ., data = training_set)

# Predicting the Test set results
y_pred = predict(regressor, newdata = test_set)
```

Application avec Python

Exécuter le script suivant dans anaconda python.

Étape 1 : Importation des bibliothèques

Dans cette première étape, nous importerons les bibliothèques nécessaires pour créer le modèle ML. Les bibliothèques numPy et matplotlib sont importées. De plus, nous avons importé la bibliothèque pandas pour l'analyse des données.

```r
import numpy as np
import matplotlib.pyplot as plt
from pandas as pd
```

Étape 2 : Importer le jeu de données

Dans cette suivante, nous utiliserons pandas pour stocker les données charger que nous allons charger à partir du fichier "50_Startups.csv" sous la forme d'un DataFrame de la bibliothèque pandas nommé " **dataset** " à l'aide de la fonction " *pd.read_csv* ".
Nous parcourons notre ensemble de données et attribuons la variable indépendante (x) aux quatre premières colonnes de notre ensemble de données, à savoir R&D Spend (*index=0*), Administration (*index=1*), Marketing Spend (*index=2*) et State (*index= 3*).

```r
dataset = pd.read_csv('50_Startups.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].valuesdataset.head(5)

>>
R&D Spend  Administration  Marketing Spend   State       Profit
165349.20  136897.80       471784.10         New York    192261.83
162597.70  151377.59       443898.53         California  191792.06
153441.51  101145.55       407934.54         Florida     191050.39
144372.41  118671.85       383199.62         New York    182901.99
142107.34  91391.77        366168.42         Florida     166187.94
```

Nous utilisons la fonction .iloc correspondante pour découper le DataFrame afin d'attribuer ces index à X. Ici, nous utilisons **[:, :-1]** qui peut être interprété comme [inclure toutes les lignes, inclure toutes les colonnes jusqu'à -1 (à l'exclusion de -1)] . En cela, -1 fait référence à la première colonne de la dernière. Ainsi, nous attribuons les 0e, 1re, 2e et 3e colonnes comme X.

Nous attribuons la dernière colonne (-1) à la variable dépendante qui est y. Nous imprimons le DataFrame pour voir si nous avons les bonnes colonnes pour nos données d'entraînement.

Étape 3 : Encodage des données catégorielles

Tant qu'il y a des nombres dans l'ensemble de données, nous pouvons facilement appliquer des calculs mathématiques sur l'ensemble de données et créer des modèles de prédiction. Dans cet ensemble de données, nous rencontrons une variable non numérique qui est « *État* ». Ceci est également appelé données *catégorielles*.

Nous encodons ces données catégorielles en utilisant une autre bibliothèque importante appelée sklearn**.** En cela, nous importons le ColumnTransformer et OneHotEncoder . Le ColumnTransformer permet à une colonne particulière du DataFrame d'être transformée séparément. Dans notre cas, nous utilisons le OneHotEncoder pour transformer notre colonne "State" (index=3) en données numériques.

Après avoir encodé les données catégorielles, nous imprimons notre DataFrame X et voyons les changements. Nous voyons qu'il y a eu une inclusion de trois nouvelles colonnes au début. Chaque colonne représente un des « États ». Par exemple, dans la première ligne, la troisième colonne représente "New York" et donc la valeur "1" dans la troisième colonne.

```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
ct = ColumnTransformer(transformers =[('encoder', OneHotEncoder(), [3])],
                       remainder = 'passthrough')
X = np.array(ct.fit_transform(X))
```

Étape 4 : Diviser l'ensemble de données en ensemble d'apprentissage et en ensemble de test

Une fois que notre ensemble de données est prêt, la prochaine tâche importante consiste à diviser notre ensemble de données en ensemble d'apprentissage et en ensemble de test. Nous procédons ainsi afin d'entraîner notre modèle avec une partie des données appelée « **ensemble d'entraînement** » et de tester les résultats de la prédiction sur un autre ensemble de données appelé « **ensemble de test** ».

Nous utilisons la fonction "train_test_split" pour diviser nos données. Ici, nous donnons le "test_size = 0,2", ce qui indique que 20 % des données constituent l'ensemble de test. Dans notre cas, 10 données de démarrage aléatoires seront choisies comme ensemble de test et 40 données de démarrage restantes seront choisies pour l'ensemble d'apprentissage.

```
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)
```

Étape 5 : Entraîner le modèle de régression linéaire multiple sur l'ensemble d'entraînement

Dans l'étape suivante, nous importons la classe " **LinearRegression** " qui va être appliquée à notre ensemble d'apprentissage. Nous attribuons une variable « **regressor** » à la classe LinearRegression. Nous utilisons ensuite « **regressor.fit** » pour ajuster l'ensemble de données d'entraînement (X_train et y_train) à cette classe LinearRegression pour que le processus d'entraînement se produise.

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

Étape 6 : Prédire les résultats de l'ensemble de test

Dans l'étape suivante, nous allons prédire le profit de l'ensemble de test à l'aide du modèle formé, nommément « régresseur ». Les valeurs réelles (bénéfices) des données de l'ensemble de test (X_test) sont stockées dans la variable y_test.

Nous utilisons ensuite la fonction « **regressor.predict** » pour prédire les valeurs de nos données de test X_test. Nous attribuons les valeurs prédites comme y_pred. Nous avons maintenant deux données, y_test (valeurs réelles) et y_pred (valeurs prédites).

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

Étape 7 : Comparaison de l'ensemble de test avec les valeurs prédites

Dans cette étape, nous imprimerons à la fois les valeurs de y_test en tant que ***valeurs réelles*** et les valeurs de y_pred en tant que ***valeurs prédites*** de chaque X_test dans un DataFrame pandas. De cette façon, nous obtenons les valeurs pour toutes les 10 données X_test.

```
df = pd.DataFrame({'Real Values':y_test, 'Predicted Values':y_pred})
df
>>
Real Values Predicted
Values 78239.91
74963.602167 182901.99 1
73144.548525 64926.08
45804.248438 105733.54 1
08530.843936 141585.52 1
27674.466487 108552.04 1
11471.421444 146121.95 1
33618.038644 105008.31 1
14655.651664 96778.92
96466.443219 97483.56
```

Dans la première ligne, les valeurs réelles ont une valeur de ***78239,91*** et les valeurs prédites ont une valeur de ***74963,60*** . Nous voyons que le modèle a prédit étroitement cette valeur et nous pouvons donc dire que notre modèle a une bonne précision.

Exercice 3 — Application de la Régression Polynomiale

Dans ces données, nous avons les deux variables indépendantes à savoir, *Position et Level* . Il existe une variable indépendante, c'est-à- *dire le salaire* . Donc, dans ce problème, nous devons former un modèle de régression polynomiale avec ces données pour comprendre la corrélation entre le niveau et le salaire des données des employés dans l'entreprise et être en mesure de prédire le salaire (variable de sortie Salary) du nouvel employé sur la base de ces données.

Application avec R

Exécuter le script suivant dans R.

```
# Polynomial Regression

# Importing the dataset
dataset = read.csv('Position_Salaries.csv')
dataset = dataset[2:3]
```

```r
# Splitting the dataset into the Training set and Test set
# # install.packages('caTools')
# library(caTools)
# set.seed(123)
# split = sample.split(dataset$Salary, SplitRatio = 2/3)
# training_set = subset(dataset, split == TRUE)
# test_set = subset(dataset, split == FALSE)

# Feature Scaling
# training_set = scale(training_set)
# test_set = scale(test_set)

# Fitting Linear Regression to the dataset
lin_reg = lm(formula = Salary ~ ., data = dataset)

# Fitting Polynomial Regression to the dataset
dataset$Level2 = dataset$Level^2
dataset$Level3 = dataset$Level^3
dataset$Level4 = dataset$Level^4
poly_reg = lm(formula = Salary ~ .,
              data = dataset)

# Visualising the Linear Regression results
# install.packages('ggplot2')
library(ggplot2)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = dataset$Level, y = predict(lin_reg, newdata = dataset)),
            colour = 'blue') +
  ggtitle('Truth or Bluff (Linear Regression)') +
  xlab('Level') +
  ylab('Salary')

# Visualising the Polynomial Regression results
# install.packages('ggplot2')
library(ggplot2)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = dataset$Level, y = predict(poly_reg, newdata = dataset)),
            colour = 'blue') +
  ggtitle('Truth or Bluff (Polynomial Regression)') +
  xlab('Level') +
  ylab('Salary')

# Visualising the Regression Model results (for higher resolution and smoother curve)
# install.packages('ggplot2')
library(ggplot2)
x_grid = seq(min(dataset$Level), max(dataset$Level), 0.1)
ggplot() +
  geom_point(aes(x = dataset$Level, y = dataset$Salary),
             colour = 'red') +
  geom_line(aes(x = x_grid, y = predict(poly_reg,
                                        newdata = data.frame(Level = x_grid,
                                                             Level2 = x_grid^2,
                                                             Level3 = x_grid^3,
                                                             Level4 = x_grid^4))),
            colour = 'blue') +
  ggtitle('Truth or Bluff (Polynomial Regression)') +
```

```
  xlab('Level') +
  ylab('Salary')

# Predicting a new result with Linear Regression
predict(lin_reg, data.frame(Level = 6.5))

# Predicting a new result with Polynomial Regression
predict(poly_reg, data.frame(Level = 6.5,
                             Level2 = 6.5^2,
                             Level3 = 6.5^3,
                             Level4 = 6.5^4))
```

Application avec Python

Exécuter le script suivant dans anaconda python.

Étape 1 : Importation des bibliothèques

Dans cette première étape, nous importerons les bibliothèques nécessaires pour créer le modèle ML. Les bibliothèques *numpy* et *matplotlib* sont importées. De plus, devons importer la bibliothèque *pandas* pour l'accès et à l'analyse des données.

```
import numpy as np
import matplotlib.pyplot as plt
from pandas as pd
```

Étape 2 : Importer le jeu de données

Dans cette étape, nous allons utiliser pandas pour stocker les données obtenues à partir de mon référentiel github et les stocker en tant que Pandas DataFrame en utilisant la fonction « pd.read_csv ».

Nous parcourons notre ensemble de données et attribuons la variable indépendante **(x)** à la deuxième colonne avec le nom de colonne « Level » et la variable dépendante **(y)** à la dernière colonne, qui est le « Salary » à prédire.

```
dataset = pd.read_csv('PositionSalaries_Data.csv')
X = dataset.iloc[:, 1:-1].values
y = dataset.iloc[:, -1].valuesdataset.head(5)

>>
Position            Level   Salary
Business Analyst    1       45000
Junior Consultant   2       50000
Senior Consultant   3       60000
Manager             4       80000
Country Manager     5       110000
```

Nous utilisons la fonction .iloc correspondante pour découper le DataFrame afin d'attribuer ces index à X et Y. En cela, *Level* est pris comme variable indépendante et est attribué à X. La variable dépendante à prédire est la dernière colonne (-1) qui est *Salary* et il est affecté à y. Nous imprimons le DataFrame " **dataset** " pour voir si nous avons les bonnes colonnes pour nos données d'entraînement.

Étape 3 : Entraîner le modèle de régression polynomiale sur l'ensemble de données

L'ensemble de données sur lequel nous utilisons a très peu de lignes et, par conséquent, nous formons l'ensemble de données pour construire le modèle de régression polynomiale. En cela, la fonction « ***PolynomialFeatures*** » est utilisée pour attribuer le degré de la ligne polynomiale que nous allons tracer. Dans ce cas, le degré est fixé à ***4*** .

La variable indépendante X est ensuite ajustée avec la classe PolynomialFeatures et est convertie en une nouvelle variable ***X_poly*** . En cela, la variable X est convertie en une nouvelle matrice X_Poly qui se compose de toutes les combinaisons polynomiales de caractéristiques de degree=4.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(X)
lin_reg = LinearRegression()
lin_reg.fit(X_poly, y)
```

La classe « ***LinearRegression »*** est également importée et est affectée à la variable ***« lin_reg »*** qui est équipée des X_poly et y pour la construction du modèle.

Étape 4 : Prédire les résultats

Dans cette étape, nous allons prédire les valeurs Salary en fonction du modèle de régression polynomiale construit. La fonction « **regressor.predict** » est utilisée pour prédire les valeurs de notre variable indépendante, X_poly. Nous attribuons les valeurs prédites comme y_pred. Nous avons maintenant deux données, y (valeurs réelles) et y_pred (valeurs prédites).

```
y_pred = lin_reg.predict(X_poly)
```

Étape 5 : Comparer les valeurs réelles avec les valeurs prédites

Dans cette étape, nous imprimerons à la fois les valeurs de y en tant que ***valeurs réelles*** et les valeurs de y_pred en tant que ***valeurs prédites*** de chaque X_test dans un Pandas DataFrame.

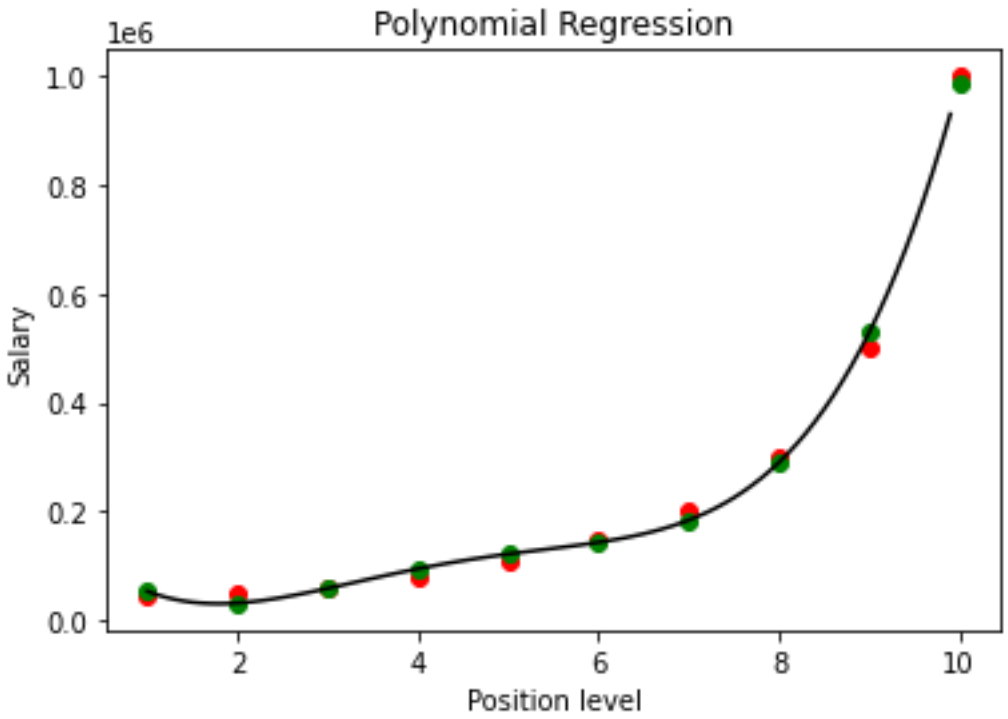
```
df = pd.DataFrame({'Real Values':y, 'Predicted Values':y_pred})
df


>>
Real Values     Predicted
Values 45000
53356.643357    50000
31759.906760    60000
58642.191142    80000
94632.867133    110000 1
21724.941725    150000 1
43275.058275    200000 1
84003.496504    300000 2
89994.172494    500000 5
28694.638695    1000000 9
```

Nous pouvons voir que le modèle a fait un excellent travail en ajustant les données et en prédisant le salaire de l'employé en fonction du niveau du poste.

Étape 6 : *Visualisation des résultats de la régression polynomiale*

Dans cette dernière étape, nous allons visualiser le modèle polynomial qui a été construit en utilisant les données données et tracer les valeurs de " ***y*** " et " ***y_pred*** " sur le graphique et analyser les résultats

```
X_grid = np.arange(min(X), max(X), 0.1)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X, y, color = 'red')
plt.scatter (X, y_pred, color = 'green')
plt.plot(X_grid, lin_reg.predict(poly_reg.fit_transform(X_grid)), color = 'black')
plt.title('Régression polynomiale')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



Modèle de régression polynomiale

Dans ce graphique, les valeurs réelles sont tracées en couleur « ***Rouge*** » et les valeurs prédites sont tracées en couleur « ***Vert*** ». La ligne de régression polynomiale qui est générée est dessinée en couleur « ***noire*** ».

Partie 2 — Application de la Régression

Exercice 1 — Application de la Régression Simple

Le fichier DataHeight.txt contient respectivement l'âge et la taille de 50 enfants. L'objectif de ce travail est de montrer qu'il existe une relation linéaire entre l'âge et la taille des enfants. Pour y parvenir, il est vous ai demandé de procéder aux étapes suivantes :

Questions :

1. Charger les données dans une variable nommée **tab**.
2. Charger le vecteur **age** dans une variable nommée **x**.
3. Charger le vecteur **height** dans une variable nommée **y**.
4. Donner la distribution de y en fonction de x (utiliser les fonctions plot et **qplot**).
5. On regardant le graphique. Peut-on dire qu'il y a une corrélation entre les deux variables. Justifiez.
6. Prouvez d'une manière statistique (rigoureuse) que x et y sont soit corrélées soit non ?
7. Quelles sont les valeurs des paramètres (θ_0 et θ_1).
8. Quelle est la valeur de l'erreur résiduelle.
9. Donnez l'équation de la droite de régression.
10. Dessiner la droite de régression.
11. Selon le modèle trouvé prédire les poids de 4 enfants ayant respectivement 3, 7, 9 et 12 ans.
12. Prédire la taille d'un enfant de 10 ans avec un intervalle de confiance de 95%.

Exercice 2 —Régression Multiple & Polynomiale

Nous voulons étudier les problèmes respiratoires d'une population de **725** individus afin de prédire les causes de ces problèmes. Nous avons à cet effet 6 variables nommées respectivement **LangCup**, **Age**, **Height**, **Smoke**, **Caesarean** représentant respectivement la capacité respiratoire, l'âge, la taille, si l'individu est fumeur ou nom et enfin s'il a une eu une naissance normale ou par césarienne.

Il est demandé d'exécuter le programme R suivant d'interpréter les résultats de chaque étape.

```
LungCapData <- read.table(file.choose(), header = T, sep="\t")
attach(LungCapData)
names(LungCapData)
class(Age)
class(Smoke)
levels(Smoke)
model1 <- lm(LungCap ~ Age + Height)
summary(model1)
cor(Age, Height, method="pearson")
confint(model1, coef.level=0.95)
model2 <- lm(LungCap ~ Age + Height + Smoke + Gender + Caesarean)
summary(model2)
plot(model2)
```

Exercice 3 — Régression Polynomiale

Cette fois-ci, nous espérons trouver une liaison entre une ou plusieurs variables de la base Boston que vous trouverez dans le package MASS. Cette base nous renseigne sur la valeur d'un logement dans la banlieue de Boston (USA). Elle contient enregistrement de variables à l'instar de :

- crim : pourcentage d'habitants criminels par ville.
- **rad** : indice d'accessibilité aux autoroutes radiales.
- **black** : proportion de noirs par 1000 individus et par ville.
- **lstat** : statut minimal de la population (pourcentage).
- **medv** : valeur médiane des habitants-propriétaires en 1000$ par unité.
- Etc

Questions :

1. Afficher les informations.
2. Afficher le type de chaque donnée.
3. Que pouvez-vous dire de ces données.
4. Diviser les données en utilisant les 400 premières observations que les données d'entraînement et le reste en tant que données de test.
5. Vérifier s'il existe une relation linéaire entre **medv** et age.
6. Dessiner le nuage de points de ces deux variables.
7. Faite la même chose avec **medv** et lstat.
8. Dessiner la droite de régression.
9. Vérifier statiquement et graphiquement si la variable medv peut être expliqué par un modèle linéaire simple par lstat.
10. Que pouvez-vous déduire ?
11. Exécutez et commenter l'instruction pairs(Boston).
12. Répéter l'instruction mais uniquement avec les première, la troisième et la septième variable.
13. utiliser l'ensemble de données de formation pour former le modèle linéaire multiple avec comme variable expliquée medv et comme variables explicatives la variable lstat et la variable age.
14. Que peut-on conclure ? Expliquez
15. utiliser l'ensemble de données de formation pour former le modèle linéaire multiple avec comme variable expliquée medv et comme variables explicatives le logarithme de la variable lstat et la variable age.
16. Que peut-on conclure ? Expliquez
17. Vérifier la relation linéaire medv entre et toutes les autres variables.
18. Que peut-on conclure ? Expliquez
19. Reconstruire le modèle linéaire avec toutes les variables sauf celles non significatives.
20. Que peut-on conclure ? Expliquez
21. Construire le modèle linéaire avec la variable medv et l'interaction entre lstat et age.
22. Vérifier si le modèle est non linéaire de degré 2, 3, 4, 5, 6 et 7.
23. Que peut-on conclure.