

Regression in Python

Dieses Notebook beinhaltet das Praktische Vorgehen bei dem Aufbau (nicht) linearer Regressionen. Hierfür werden die, in der Arbeit beschriebenen, Pakete NumPy, Pandas, Mathplotlib und SciPy bzw. die funktion `curve_fit` von dem SciPy Unterpaket `SciPy.optimize` genutzt. Anzumerken ist, dass die Schritte der Datensammlung und Aufbereitung hier nicht behandelt werden, da dies den Rahmen dieser Arbeit überschreitet. Im folgenden wird ein Dataset genutzt, welches frei Verfügbar auf der Website Kaggle aufzufinden ist. Anzumerken ist, das Grundsätzlich alle Erkenntnisse mit dem jeweils notwendigem Fachwissen begutachtet werden müssen, um nicht ehleleitete Schlüsse zu ziehen.

```
In [ ]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from scipy.stats import linregress
```

Welche Daten brauchen wir?

Um dieses Dataset zu nutzen, müssen wir es zu erst importieren. Dies geschieht mittels des Pandas Paket. Die `Pandas.read...()` funktionen bieten hierfür eine gute leicht nutzbare Möglichkeit

```
In [ ]: df = pd.read_csv('longley.csv')
df.head()
```

```
Out[ ]:
```

| | 83 | 234.289 | 235.6 | 159 | 107.608 | 1947 | 60.323 |
|---|------|---------|-------|-------|---------|------|--------|
| 0 | 88.5 | 259.426 | 232.5 | 145.6 | 108.632 | 1948 | 61.122 |
| 1 | 88.2 | 258.054 | 368.2 | 161.6 | 109.773 | 1949 | 60.171 |
| 2 | 89.5 | 284.599 | 335.1 | 165.0 | 110.929 | 1950 | 61.187 |
| 3 | 96.2 | 328.975 | 209.9 | 309.9 | 112.075 | 1951 | 63.221 |
| 4 | 98.1 | 346.999 | 193.2 | 359.4 | 113.270 | 1952 | 63.639 |

Nun, da wir die Daten auslesen können, ist es wichtig sich ein Bild von der abhängigen und der einen oder den mehreren unabhängigen Variablen zu machen.

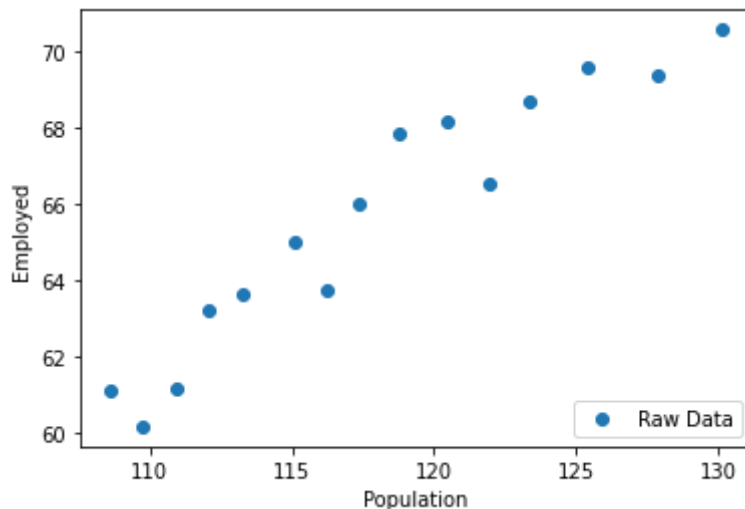
Wie sehen unsere Daten aus?

In diesem Teil untersuchen wir die Daten nun, um daraufhin zu entscheiden, welche Art von Funktion nötig ist, und um für die folgenden Schätzungen eine bessere Entscheidungsgrundlage zu schaffen. Hierzu plotten wir die Datenpunkte.

```
In [ ]: df = pd.read_csv('longley.csv')
data = df.values
x, y = data[:, 4], data[:, -1]
```

```
plt.scatter(x, y, label='Raw Data')
plt.xlabel('Population')
plt.ylabel('Employed')
plt.legend(loc='lower right')
```

Out []: <matplotlib.legend.Legend at 0x1a929abec20>



Diese Datenpunkte lassen einen Linearen Zusammenhang vermuten.

Linear Regression

```
In [ ]: linregress = linregress(x,y)
linregress
```

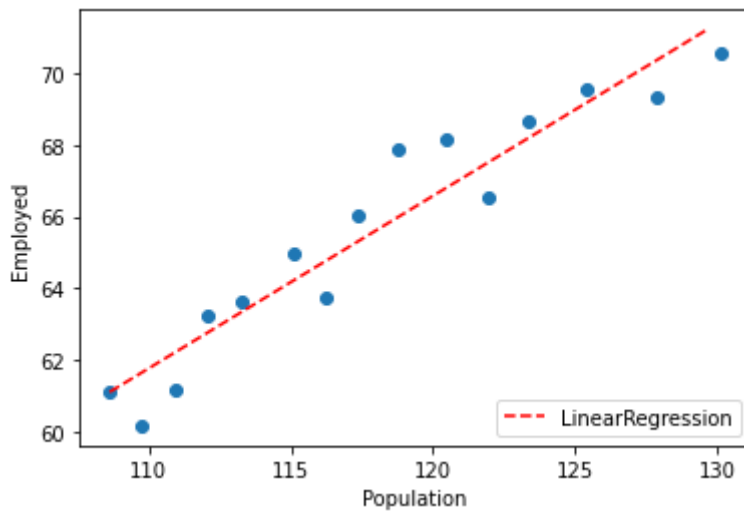
Out []: LinregressResult(slope=0.48093826579216153, intercept=8.861512409820172, rvalue=0.9538032648633806, pvalue=3.673093153978152e-08, stderr=0.04201505240315141, intercept_stderr=4.9684543221991255)

```
In [ ]: def linear(x):
        return linregress.slope*x + linregress.intercept

x_monotonic = np.arange(min(x),max(x),1)
y_fit = linear(x_monotonic)

plt.plot(x_monotonic,y_fit,'--',color='red',label='LinearRegression')
plt.legend(loc='lower right')
plt.scatter(x, y, label='Raw Data')
plt.xlabel('Population')
plt.ylabel('Employed')
```

Out []: Text(0, 0.5, 'Employed')



How to use: Curve Fitting

Nun da wir entschieden haben, einen linearen Zusammenhang zu vermuten, ist es an der Zeit die nötigen Parameter für das CurveFitting zu definieren. Mindestens gebraucht werden hierbei die Daten der abhängigen und unabhängigen Variablen wie auch die jeweils gewählte Funktion.

Die Funktion `linear_fit` hat als Parameter an erster Stelle die unabhängige Variable `x` und als darauffolgende Parameter die Parameter der Funktion die optimiert werden sollen. Nun, da die mindestanforderungen für ein CurveFitting erfüllt sind, können wir nun die Funktion nutzen.

```
In [ ]: def linear_fit(x,m,b):
        return m*x + b
        popt, pcov = curve_fit(linear_fit, x, y)
```

Die Funktion `curve_fit` gibt im normalfall 2 werte zurück. `Popt` und `pcov`. `Popt` beinhaltet ein Array bestehend aus den optimalen Werten für die Parameter der `linear_fit` Funktion.

```
In [ ]: m, b = popt
        print(f"m: {m} & b: {b}")

m: 0.4809382705278997 & b: 8.861511850631778
m: 0.4809382705278997 & b: 8.861511850631778
```

slope= 0.48093826579216153, intercept= 8.861512409820172

`Pcov` ist die vermutete Covarianz von `popt`, aus welcher man z.B. die Standardabweichung bzw. den Fehler ablesen kann. Dies ist hilfreich um das Modell zu bewerten, jedoch wird in weiteren Teilen dieses Notebooks noch auf die Fehleranalyse eingegangen.

Die optimalen Parameter kann man nun für `m` respektive `b` einsetzen und die lineare Regressionsgerade ist fertig.

```
In [ ]: print(f"y = {m.round(2)}x + {b.round(2)}")

y = 0.48x + 8.86
```

Vermutung aufstellen

Beim Curve Fitting ist es häufig sinnvoll der Funktion `curve_fit()` erste Schätzungen für die Parameter abzugeben. Diese werden daraufhin von der Funktion mit einbezogen und helfen dabei bessere Fits für die Parameter zu finden. Hierbei ist zu beachten, dass die Qualität der Vermutung die Qualität der von `curve_fit()` aufgestellten Parameter beeinflusst. Der Parameter `p0` der Funktion `curve_fit` nimmt hierbei Beta, also den Vektor an Parametern an. Hierbei muss die Reihenfolge und Anzahl der Parameter die selbe sein, wie die in der definierten Funktion.

Anmerkung: Wenn keine Vermutung aufgestellt wird wird für jeden parameter der Wert 1 vermutet.

```
popt, pcov = curve_fit(linear_fit, x, y) == popt, pcov =
curve_fit(linear_fit, x, y, p0=[1, 1])
```

Aus den scatter Plot kann man vermuten, dass die Population Anzahl doppelt so schnell steigt, wie die der Mitarbeiter, da von 110 bis 120 Population die Werte der Mitarbeiter von ca 61 auf ca 66 steigen. Unsere Vermutung für m wäre daher 0.5. Aus dem Abschnitt, den wir in den Daten haben kann man nicht leicht auf den Y-Achsen Abschnitt (also b) kommen. Aber wenn die erste Annahme von $m = 0.5$ genutzt wird ergibt sich schnell, dass der Y-Achsenabschnitt bei ca. 6 sein muss (Wenn man von 110 Population für alle 10 an Population weniger 5 an Employee weniger geht).

```
In [ ]: popt, pcov = curve_fit(linear_fit, x, y, p0=[0.5, 6])
print("Optimale Parameter")
print(f"m: {m} & b: {b}")
print("Gerade:")
print(f"y = {m.round(2)}x + {b.round(2)}")
```

```
Optimale Parameter
m: 0.4809382705278997 & b: 8.861511850631778
Gerade:
y = 0.48x + 8.86
```

Nicht lineare Regression

Es gibt bei nicht linearen Regressionen nicht nur die eine, wie es bei Linearen immer eine Form von $y = mx + b$ ist. Zu den Formen nicht linearer Regressionen gehören z.B. Exponentialfunktion, Logarithmusfunktionen und andere (aus dem Teil von den anderen übernehmen). Wie für verschiedene nicht lineare Funktionen die Bestimmung in der Theorie aussieht wurde in vorherigen Teilen dieser Arbeit definiert, hier wird darauf eingegangen, wie nicht lineare Regressionen praktisch anzuwenden sind und an einem Beispiel erläutert.

Um mit SciPys `curve_fit()` Funktionen einen nicht Linearen Zusammenhang darzustellen gibt es eine naheliegende Lösung. Wie in Vorherigen Teilen dieser Arbeit beschrieben brauchen wir um diese Funktion zu nutzen eine Funktion, welche den Zusammenhang zwischen den Daten beschreiben soll. Diese Funktion muss aber nicht linear sein, und kann jeden

beliebigen Zusammenhang darstellen. Im folgenden wird jede besprochene Form einer nicht linearen Regression Vorgestellt.

Exponentielle Regression

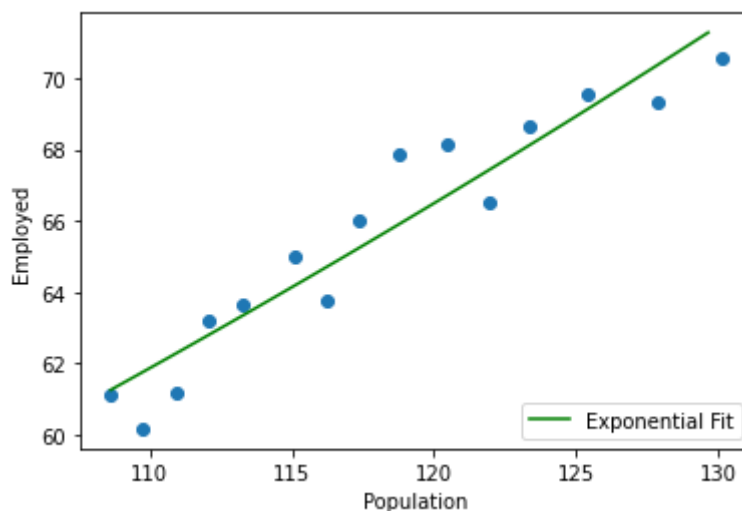
```
In [ ]: def exponential_func(x,a,b):
        return a * np.power(b,x)
```

```
In [ ]: popt, pcov = curve_fit(exponential_func, x, y)
a, b = popt
print("Optimale Parameter")
print(f"a: {a.round(2)} & b: {b.round(2)}")
print("Funktion:")
print(f"f(x) = {a.round(2)} * {b.round(2)}\u00E3")
```

```
Optimale Parameter
a: 27.97 & b: 1.01
Funktion:
f(x) = 27.97 * 1.01x
```

```
In [ ]: x_monotonic = np.arange(min(x),max(x),1)
#y_fit = linear_fit(x_monotonic)
#plt.plot(x_monotonic,y_fit,'--',color='red',label='Linear Fit')
y_fit = exponential_func(x_monotonic,*popt)
plt.plot(x_monotonic,y_fit,'-',color='green',label='Exponential Fit')
plt.legend(loc='lower right')
plt.scatter(x, y, label='Raw Data')
plt.xlabel('Population')
plt.ylabel('Employed')
```

```
Out[ ]: Text(0, 0.5, 'Employed')
```



```
In [ ]: x_monotonic = np.arange(min(x),200,1)
y_fit = linear(x_monotonic)
plt.plot(x_monotonic,y_fit,'--',color='red',label='Linear Fit')
y_fit = exponential_func(x_monotonic,*popt)
plt.plot(x_monotonic,y_fit,'-',color='green',label='Exponential Fit')
plt.legend(loc='lower right')
plt.scatter(x, y, label='Raw Data')
plt.xlabel('Population')
plt.ylabel('Employed')
```

```
Out[ ]: Text(0, 0.5, 'Employed')
```

