Merkblatt Lineare Regression

Lineare Regression

[14]:
```
import numpy as np
import pandas as pd
import seaborn as sns

from ipywidgets import interact
import ipywidgets as widgets
```

Die Lineare Regression ist dazu geeignet, Daten mit Hilfe einer Geraden zu beschreiben, mit deren Hilfe man Voraussagen zu nicht vorhandenen Daten machen kann, zum Beispiel: Wie viel kostet ein Diamant, wenn er 10 Karat hat?

Um besser zu erklären, was die Lineare Regression macht, ist nachfolgendes Beispiel gedacht. Es wird immer der quadrierte Abstand aus allen Punkten zur Geraden errechnet. Damit die Summe aller Abstände den kleinsten Wert annimmt, nehmen wir an, die Gerade würde im vorstehenden Plot auf der Y-Achse auf

1(Blau) stehen, daraus ergeben sich folgende Werte: 0*0 + 4*4 + 4*4 + 0*0 = 32

2(Orange) stehen, daraus ergeben sich folgende Werte: 1*1 + 3*3 + 3*3 + 1*1 = 20

3(Grün) stehen, daraus ergeben sich folgende Werte: 2*2 + 2*2 + 2*2 + 2*2 = 16

4(Rot) stehen, daraus ergeben sich folgende Werte: 3*3 + 1*1 + 1*1 + 3\3 = 20

5(Lila) stehen, daraus ergeben sich folgende Werte: 4*4 + 0*0 + 0*0 + 4*4 = 32

Daraus ergibt sich, dass die Funktion der Linearen Regression für diese Funktion y = 3 ist. Dieser Versatz von der `y = 0`-Achse nennt sich auch Intercept, der Intercept ist hier also 3.

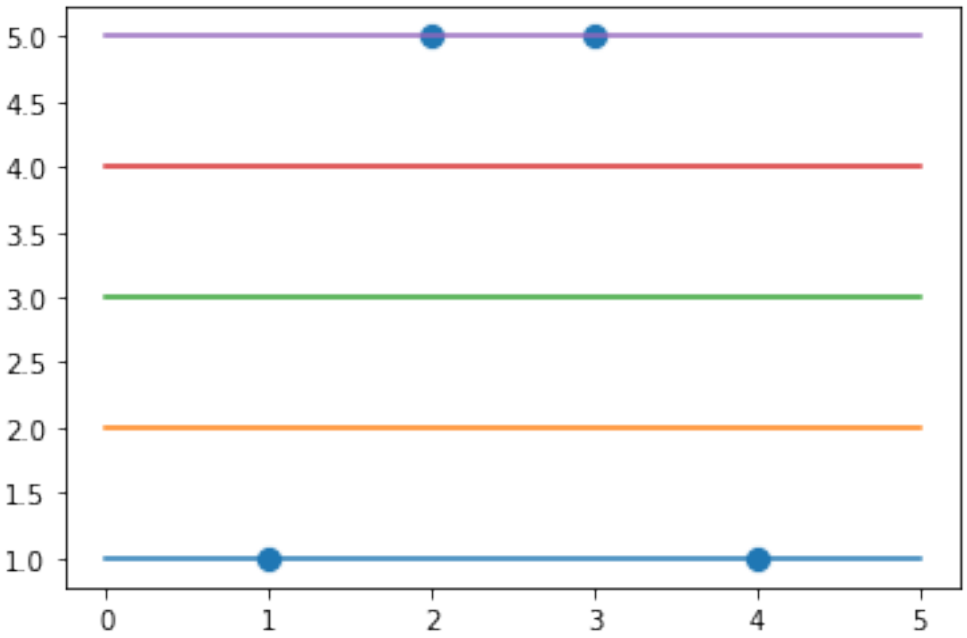
Wichtig: In der Praxis hat diese Gerade natürlich auch eine Steigung (Koeffizient). Dieses Beispiel habe ich aber so gewählt, dass die Steigung gleich 0 ist, um deutlicher zu machen, warum wir den quadierten Abstand verwenden.

[12]:
```
xs = [1, 2, 3, 4]
ys = [1, 5, 5, 1]

ax = sns.scatterplot(xs, ys , s = 100)
sns.lineplot([0,5], 1,)
sns.lineplot([0,5], 2,)
sns.lineplot([0,5], 3,)
sns.lineplot([0,5], 4,)
sns.lineplot([0,5], 5,)
#    sns.lineplot([0, 5], [x, x], ax = ax, color = "red")
#    return x
```

```
#interact(f, x = widgets.FloatSlider(min = 1, max = 5, step = 0.1, value=3));
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0xffff5f81fbb0>

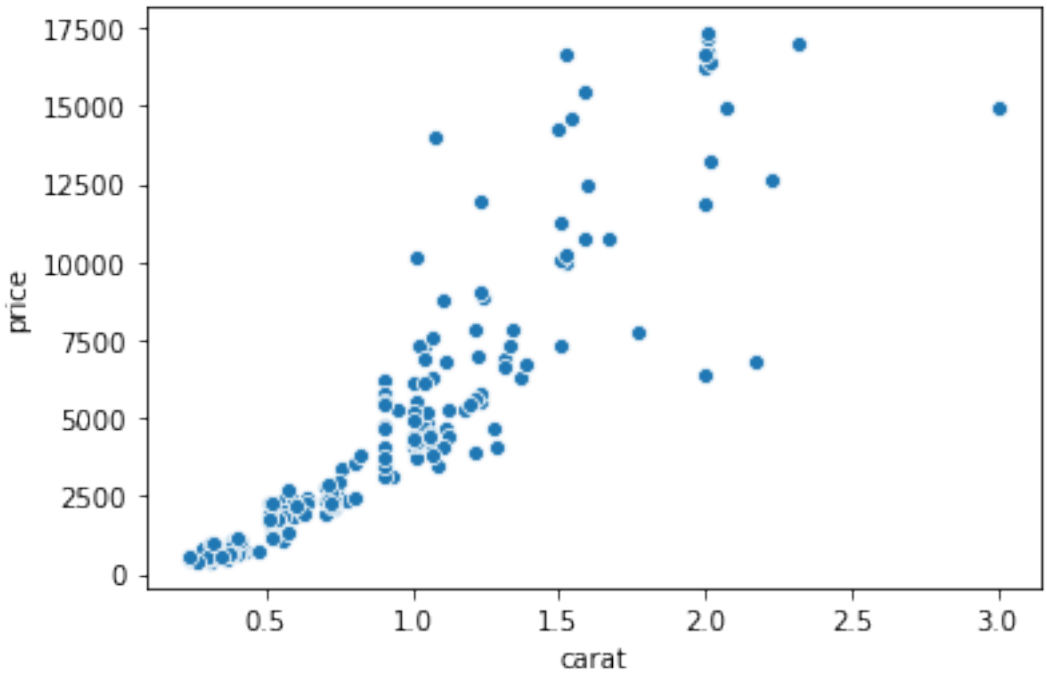


[13]:
```
df = pd.read_csv("./diamonds.csv")
df.head()
```

[13]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

[4]:
```
sns.scatterplot(x = "carat", y = "price", data = df.sample(200));
```



Um das Modell zu trainieren, müssen wir zuerst die X-Daten sammeln. Hierbei ist es wichtig, dass die Shape vom Numpy-Array korrekt ist: In der ersten Dimension steht die Anzahl der Einträge (hier: 53940 Einträge), in der 2. Dimension die Anzahl der Features, auf dessen Basis wir vorhersagen machen möchten (hier: 1 Feature, "carat").

[5]:
```
xs = df["carat"].to_numpy().reshape(-1, 1)
# alternativ: xs = df[["carat"]].to_numpy()
print(xs.shape)
```

```
(53940, 1)
```

Bei `ys` darf nur ein Wert pro Zeile stehen, hier `price`. Dies ist der Wert, der anhand der Werte der Features ermittelt werden soll.

[6]:
```
ys = df["price"].to_numpy()
print(ys.shape)
```

```
(53940,)
```

Preis eines Diamanten = Model.coef_ * Gewicht in Karat + model.intercept_

Mithilfe von der `LinearRegression`-Funktion, die von `sklearn.linear_model` importiert werden, können nun diese NumpyArrays in die Geradengleichung y = a*x + b umgewandelt werden, wobei a die Steigung für jedes Feature (hier gibtz es nur 1, daher Array der Größe 1) angibt (model.coef_), und b den Schnittpunkt mit der y-Achse (model.intercept_) angibt.

[7]:
```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(xs, ys)
print(model.coef_)
```

```
print(model.intercept_)
```

```
[7756.42561797]
-2256.3605800468918
```

Die folgenden zwei Funktionen errechnen den erwarteten Preis aufgrund der angegebenen Karatzahl.

[8]:
```
def get_price(carat):
    return model.coef_ * carat + model.intercept_
print(get_price(10))
```

```
[75307.89559966]
```

[9]:
```
model.predict([
    [10],
    [5],
    [1]
])
```

[9]:
```
array([75307.89559966, 36525.7675098 ,  5500.06503792])
```

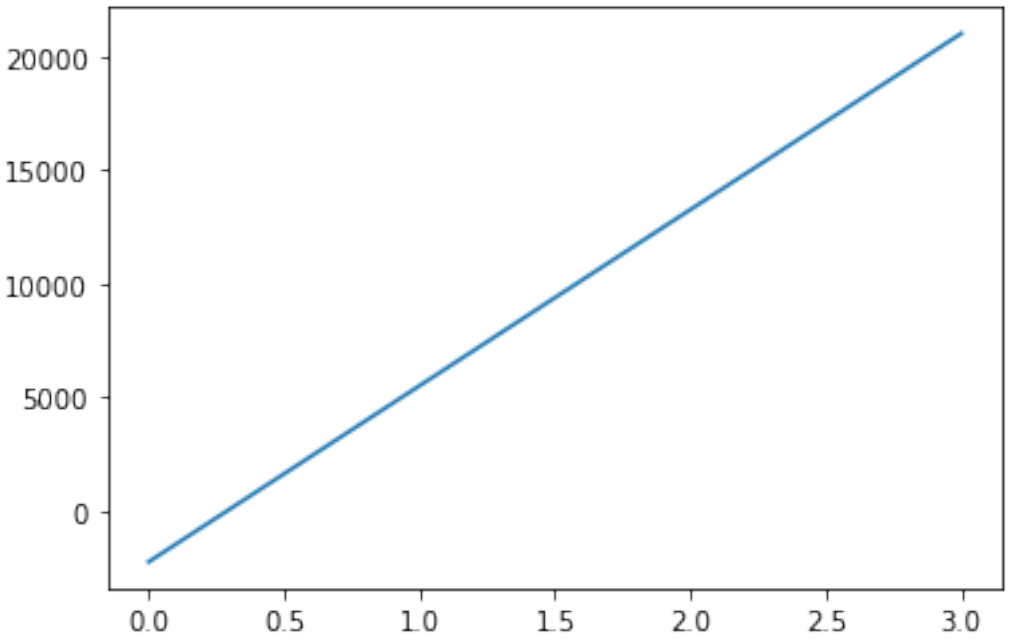
So wird mit Hilfe der Gleichung eine Gerade erstellt, dabei wird der Wertebereich durch `x_pred` der x_Achse angegeben (hier 0, 3). Auf y_pred wird die obrige Formel eingefügt, als letztes wird `x_pred` und `y_pred` als jeweils x und y in eine `sns.lineplot` Funktion eingefügt

[10]:
```
x_pred = np.array([0, 3])

y_pred = model.predict(x_pred.reshape(-1, 1))

sns.lineplot(x = x_pred, y = y_pred)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0xffff954cb100>
```



Um sowohl die einzelnen Punkte, als auch die Gerade in einen Diagramm anzuzeigen, wird die oben genannte Formel als `ax` abgekürzt und in das Scatterplot unter `ax` eingetragen. Hier wurde zwecks Anschauung noch die Gerade rot eingefärbt.

[11]:
```
ax = sns.lineplot(x = x_pred, y = y_pred, color = "red")
sns.scatterplot(x = "carat", y = "price", data = df.sample(200), ax = ax)
```

[11]: `<matplotlib.axes._subplots.AxesSubplot at 0xffff94941ee0>`