Logistische Regression

Die Logistische Regression ist ein Regressionsverfahren, das dann Anwendung findet, wenr zwei Zustände annehmen kann, also z.B. *0* oder *1*, *True* oder *False* etc. Ein wichtiges Anwer Maintenance. Obwohl es es sich um ein Regressionsverfahren handelt, wird die Logistische Klassifikationsmodell verwendet, indem ein Schwellenwert (Threshold, Cut-Off) definiert wirc sagt das Modell *1* oder *True* voraus, ansonsten *0* oder *False*.

Beispiel 1

Wir erstellen einen einfachen Beispieldatensatz mit Daten über die Anzahl der Schuljahre ur Führungskraft sind oder nicht:

In [1]:
```
import pandas as pd

df = pd.DataFrame({"Schuljahre":[8,9,13,13,8,10,10,9,10,9,11,8,15,13,10],
                   "IstFuehrungskraft": [0,0,1,1,0,1,0,0,0,0,1,0,1,0,0]})



df
```

	Schuljahre	IstFuehrungskraft
0	8	0
1	9	0
2	13	1
3	13	1
4	8	0
5	10	1
6	10	0
7	9	0
8	10	0
9	9	0
10	11	1
11	8	0
12	15	1
13	13	0
14	10	0

Wir erstellen ein *glm*-Objekt (Generalized Linear Model). Dass es sich um eine Logistische F wir der Funktion durch die Angabe *family=sm.families.Binomial* mit.

In [2]:

```
import statsmodels.formula.api as smf
import statsmodels.api as sm

model = smf.glm("IstFuehrungskraft~Schuljahre", data=df, family=sm.families.Binomial())
model.params
```

```
Intercept    -10.695596
Schuljahre     0.933998
dtype: float64
```

β_0 ist also -1.252033, β_1 ist 0.152439. Wir können somit die Wahrscheinlichkeit, dass eine F
verfügt, wie folgt berechnen:

$$\frac{1}{1+e^{10.695596-0.933998x}} = 0,92$$

Oder wir verwenden die Funktion *predict*:

In [3]:

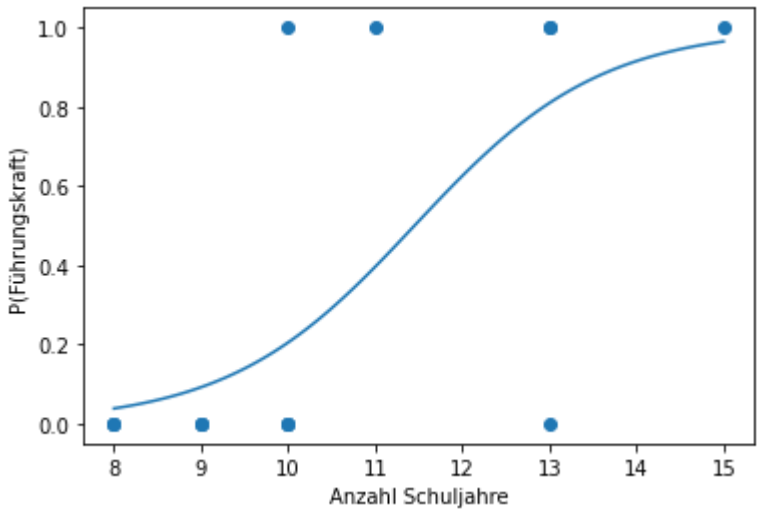
```
model.predict(pd.DataFrame({"Schuljahre":[14]}))
```

```
0    0.915318
dtype: float64
```

Plotten wir noch die Funktion:

In [4]:

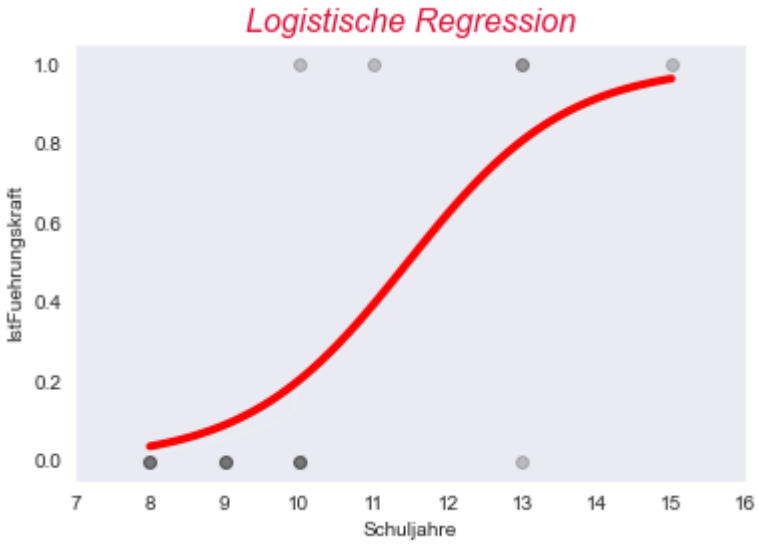
```
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

x = np.linspace(df.Schuljahre.min(), df.Schuljahre.max(), 100)
y = model.predict(pd.DataFrame({"Schuljahre":x}))

plt.scatter(df.Schuljahre, df.IstFuehrungskraft)
plt.plot(x,y)
plt.xlabel("Anzahl Schuljahre")
plt.ylabel("P(Führungskraft)")
plt.show()
```



Mit *seaborn* kann auch direkt ein Regressionsplot erstellt werden. Dazu setzt man die Optior

In [5]:

```
import seaborn as sns
df = pd.DataFrame({"Schuljahre":[8,9,13,13,8,10,10,9,10,9,11,8,15,13,10],
                   "IstFuehrungskraft": [0,0,1,1,0,1,0,0,0,0,1,0,1,0,0]})
sns.set_style("dark")
plt.xlim(7, 16)
sns.regplot(x="Schuljahre", y="IstFuehrungskraft", data=df, logistic=True, ci=False,
            line_kws={"color":"red", "lw":4},
            scatter_kws={"alpha":0.2, "color":"k"})
plt.title("Logistische Regression", color="crimson", fontsize=16, fontstyle="italic")
#plt.ylabel(r"$p_{Führungskraft}$")
plt.show()
```



In [6]:

```
model.predict(pd.DataFrame({"Schuljahre":[8]}))
```

```
0    0.038287
dtype: float64
```

Beispiel 2:

Ein weiteres "ausgedachtes" Beispiel: Wahrscheinlichkeit, an Lungenkrebs zu erkranken in A
Zigratten / Woche.

In [7]:

```
import numpy as np
X = np.array([[0, "Nein"],
              [10, "Nein"],
              [60, "Ja"],
              [90, "Ja"]])
n = X.shape[0] # Anzahl Beobachtungen
print(X)
```

```
[['0' 'Nein']
 ['10' 'Nein']
 ['60' 'Ja']
 ['90' 'Ja']]
```

Wir erstellen das Modell. Hier verwenden wir die Klasse *LogisticRegression* aus dem Packa

In [8]:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression().fit(X[:,0].reshape(-1,1), X[:,1])
print("Beta 0 = ", model.intercept_)
print("Beta 1 = ", model.coef_)
```

```
Beta 0 =  [7.74782809]
Beta 1 =  [[-0.21987113]]
```

In [9]:

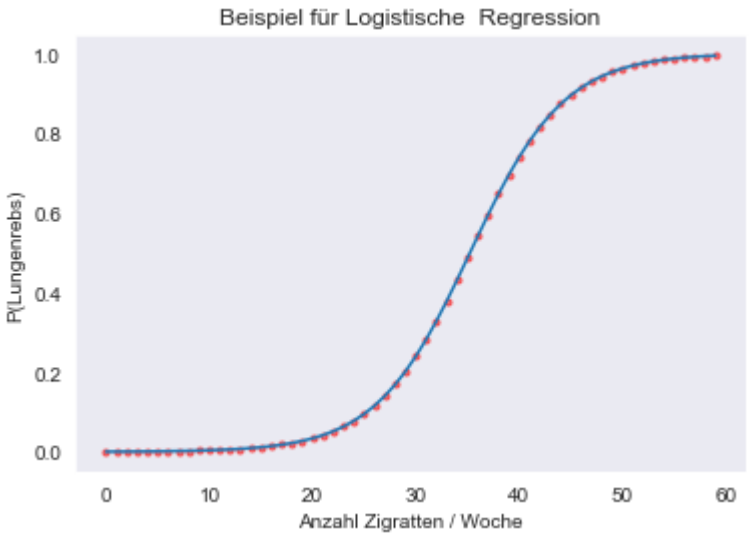
```
# Beispiele für Vorhersage: Wahrscheinlichkeiten für einige Beispieldaten.
zigaretten_tag = np.array([0, 3, 40, 50])
print(model.predict(zigaretten_tag.reshape(-1,1)))
```

```
['Nein' 'Nein' 'Ja' 'Ja']
```

Wir ploten die Sigmoid-Funktion.

In [10]:

```
%matplotlib inline
import matplotlib.pyplot as plt

x = np.arange(60)
y = model.predict_proba(x.reshape(-1,1))[:,0]

plt.scatter(x,y, c="red", s=10, alpha=.5)
plt.plot(x,y)
plt.title("Beispiel für Logistische  Regression")
plt.xlabel("Anzahl Zigratten / Woche")
plt.ylabel("P(Lungenrebs)")
plt.show()
```



Beispiel 3: Shuttle (Challanger-Katastrophe)

Vorhersage, ob die verbaute Gummidichtung (O-Ring) bei der damals herrschenden Temper
Datensatz *shuttle.csv* und führen eine Logistische Regression durch.

In [11]:

```
import pandas as pd
import statsmodels.api as sms
import statsmodels.formula.api as sfm

url = "https://raw.githubusercontent.com/troescherw/datasets/master/shuttle.csv"
shuttle = pd.read_csv(url)

model = sfm.glm("r~temperature", data=shuttle, family=sms.families.Binomial()).fit()
model.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	r	**No. Observations:**	23
Model:	GLM	**Df Residuals:**	21
Model Family:	Binomial	**Df Model:**	1
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-10.158
Date:	Fri, 31 Dec 2021	**Deviance:**	20.315
Time:	10:43:22	**Pearson chi2:**	23.2
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	15.0429	7.379	2.039	0.041	0.581	29.505
temperature	-0.2322	0.108	-2.145	0.032	-0.444	-0.020

Am Tag es Unglücks herrschten angeblich 31 Grad Fahrenheit (-0,6 Grad Celsius). Mit welch Gummidichtung nicht funktionsfähig?

In [12]:

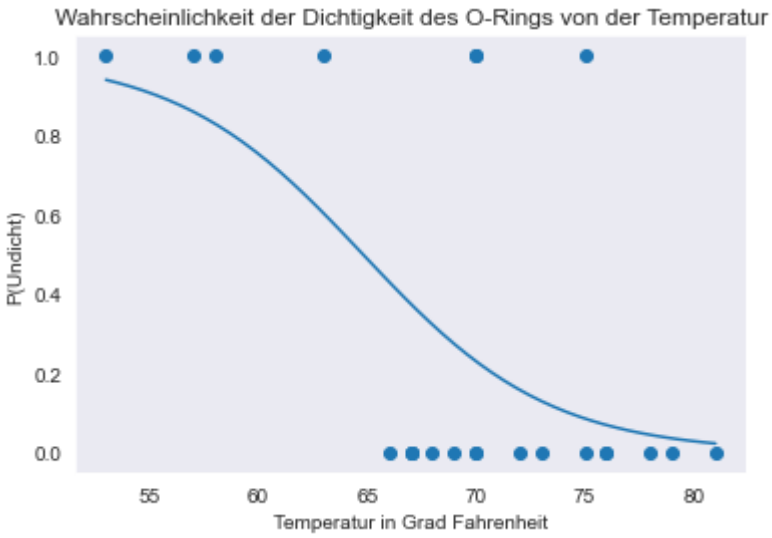
```
model.predict(pd.DataFrame({"temperature":[31]}))
```

```
0    0.999609
dtype: float64
```

Erstellen wir noch einen Plot:

In [13]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(shuttle.temperature.min(), shuttle.temperature.max(), 50)
y = model.predict(pd.DataFrame({"temperature": x}))

plt.scatter(shuttle.temperature, shuttle.r)
plt.plot(x,y)
plt.title("Wahrscheinlichkeit der Dichtigkeit des O-Rings von der Temperatur")
plt.xlabel("Temperatur in Grad Fahrenheit")
plt.ylabel("P(Undicht)")
plt.show()
```



Beispiel 4: TITANIC

Überlebenswahrscheinlicheit auf der Titanic in Abhängigkeit von Geschlecht, Alter und Klass

In [14]:

```
import pandas as pd

url = "https://raw.githubusercontent.com/troescherw/datasets/master/titanic.csv"
titanic = pd.read_csv(url)
titanic
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0
...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

891 rows × 12 columns

Wir wollen die Überlebenswahrscheinlichkeit (*Survived*) in Abhängigkeit von Alter (*Age*), Ges
(*Pclass*) vorhersagen. Wir überprüfen zuersat, ob es NaNs im Datensatz gibt:

In [15]:

```
titanic.isna().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Bei *Age* gibt es 177 NaNs! Wir ersetzen diese durch den Mittelwert der Altersangaben.

In [16]:

```
mean_age = titanic.Age.mean()
titanic.Age = titanic.Age.fillna(mean_age)
```

Nun teilen wir den Datensatz in einen Trainings- und Testdatensatz auf.70% verwenden wir f
Testen.

In [17]:

```
from sklearn.model_selection import train_test_split
X = titanic[["Survived", "Age", "Sex", "Pclass"]]
y = titanic.Survived

X_train, X_test, y_train, y_test = train_test_split(X,y,shuffle=True, random_state=42,
```

Nun erstellen wir das Vorhersagemodell, erstellen anhand der Test-Daten eine Prognose un
Confusion-Matrix dar. Beim Geschlecht und bei der Klasse handelt es sich um kategoriale V
Formel diese Kategorien in Klammern hinter einem *C*.

In [18]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

model = smf.glm("Survived~Age+C(Sex)+C(Pclass)", data=X_train, family=sm.families.Binor
model.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	Survived	**No. Observations:**	623
Model:	GLM	**Df Residuals:**	618
Model Family:	Binomial	**Df Model:**	4
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-288.41
Date:	Fri, 31 Dec 2021	**Deviance:**	576.83
Time:	10:43:23	**Pearson chi2:**	650.
No. Iterations:	5		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	3.0911	0.420	7.358	0.000	2.268	3.914
C(Sex)[T.male]	-2.5216	0.219	-11.525	0.000	-2.950	-2.093
C(Pclass)[T.2]	-0.7114	0.303	-2.350	0.019	-1.305	-0.118
C(Pclass)[T.3]	-2.0421	0.283	-7.205	0.000	-2.598	-1.487
Age	-0.0288	0.009	-3.327	0.001	-0.046	-0.012

Die Vorhersage liefert uns die Wahrscheinlichkeiten. Wir setzen einen Cut-Off bei 0,5: Bei ei Überlebenswahrscheinlichkeit >0,5 setzen wir eine *1* (Survived), ansonsten eine *0*.

In [19]:

```
from sklearn.metrics import classification_report

pred = model.predict(X_test)
pred_class =pd.Series([1 if x >0.5 else 0 for x in pred])

print(classification_report(y_test, pred_class))
```

```
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       157
           1       0.80      0.70      0.75       111

    accuracy                           0.81       268
   macro avg       0.81      0.79      0.80       268
weighted avg       0.81      0.81      0.80       268
```

Wir erreichen also eine **Accuracy** von 81%.

Sagen wir die Überlebenswahrscheinlichkeit für einen männlichen Passagier, 20 Jahre in de

In [20]:

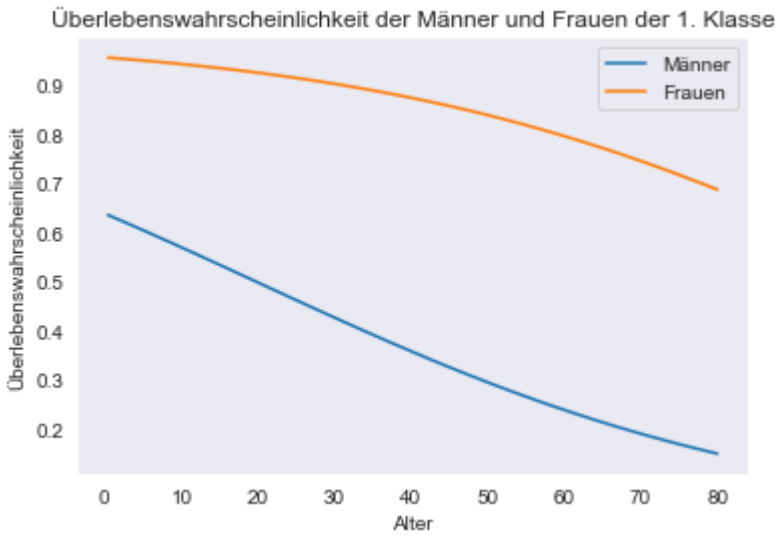
```
model.predict(pd.DataFrame({"Age": [20], "Sex":["male"], "Pclass" : 3}))
```

```
0    0.114215
dtype: float64
```

Erstellen wir noch einen Plot mit der Überlebenswahrscheinlichkeit in Abhängigkeit von Alter der 1. Klasse:

In [21]:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(titanic.Age.min(), titanic.Age.max(), 50)
y_male = model.predict(pd.DataFrame({"Sex":"male", "Age":x, "Pclass": 1}))
y_female = model.predict(pd.DataFrame({"Sex":"female", "Age":x, "Pclass": 1}))

plt.plot(x, y_male, label="Männer")
plt.plot(x, y_female, label="Frauen")
plt.legend()#
plt.xlabel("Alter")
plt.ylabel("Überlebenswahrscheinlichkeit")
plt.title("Überlebenswahrscheinlichkeit der Männer und Frauen der 1. Klasse")
plt.show()
```



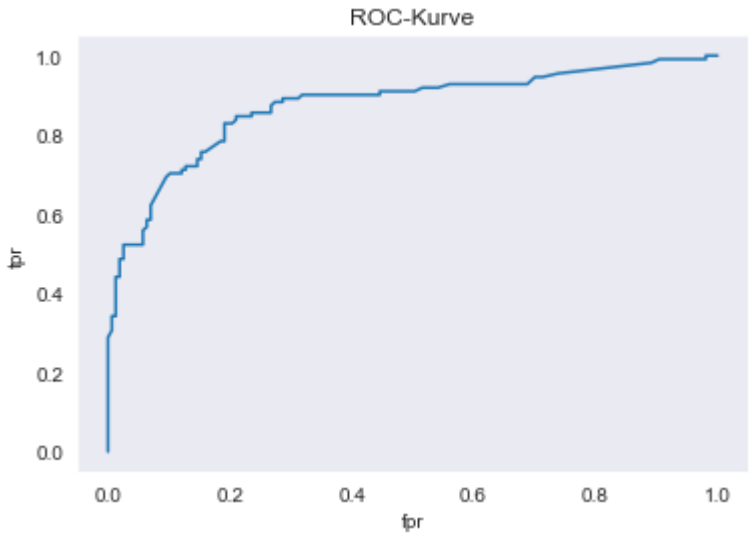
ROC-Kurve

Erstellen wir noch eine ROC-Kurve für unser Modell und geben die AUC aus.

In [22]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(y_test, pred)

plt.plot(fpr, tpr)
plt.title("ROC-Kurve")
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.plot()

print("AUC = ", roc_auc_score(y_test, pred_class))
```

```
AUC =  0.7908417972112239
```



Beispiel 5: Vorhersage einer Herzkrankheit

Ein Beispiel aus der Medizin! Wir wollen anhand des Datensatzes *heartdisease.csv* mit den vorhersagen, ob eine Person an einer Herzkrankheit erkranken wird:

Beschreibung der Features:

Feature	Bedeutung
age	age in years
sex	sex (1 = male; 0 = female)
cp	chest pain type
	Value 1 : typical angina
	Value 2 : atypical angina
	Value 3 : non-anginal pain
	Value 4 : asymptomatic
trestbps	resting blood pressure (in mm Hg on admission to the hospital)
chol	serum cholestoral in mg/dl
fbs	(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
restecg	resting electrocardiographic results
	Value 0: normal
	Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
	Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
thalach	maximum heart rate achieved
exang	exercise induced angina (1 = yes; 0 = no)
oldpeak	ST depression induced by exercise relative to rest
slope	the slope of the peak exercise ST segment
	Value 1: upsloping
	Value 2: flat
	Value 3: downsloping
ca	number of major vessels (0-3) colored by flourosopy
thal	3 = normal; 6 = fixed defect; 7 = reversable defect
hd	**diagnosis of heart disease**

Wir laden den Datensatz in ein Pandas DataFrame. Da das Feature *hd* nicht nur angibt, ob, Herzerkrankung vorliegt, setzen wir alle Werte >=1 auf 1.

In [23]:

```
import pandas as pd
url = "https://raw.githubusercontent.com/troescherw/datasets/master/heartdisease.csv"
headers = ["age","sex","cp","trestbps","chol","fbs","restecg","thalach","exang","oldpea
df = pd.read_csv(url, header=None)
df.columns=headers
df.status = ["krank" if x>=1 else "gesund" for x in df.status]
print(df.shape)
df.head()
```

```
(303, 14)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	status
0	63.0	1.0	1.0	145.0	233.0	1.0	2.0	150.0	0.0	2.3	3.0	0.0	6.0	gesund
1	67.0	1.0	4.0	160.0	286.0	0.0	2.0	108.0	1.0	1.5	2.0	3.0	3.0	krank
2	67.0	1.0	4.0	120.0	229.0	0.0	2.0	129.0	1.0	2.6	2.0	2.0	7.0	krank
3	37.0	1.0	3.0	130.0	250.0	0.0	0.0	187.0	0.0	3.5	3.0	0.0	3.0	gesund
4	41.0	0.0	2.0	130.0	204.0	0.0	2.0	172.0	0.0	1.4	1.0	0.0	3.0	gesund

In [24]:

```
from sklearn.model_selection import train_test_split

X = df
y = df.status

# Trainings- und Testdaten
X_train, X_test, y_train, y_test = train_test_split(X,y,shuffle=True, random_state=0)
```

In [25]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf
from sklearn.metrics import classification_report

# Modell erstellen
# cp, restecg und slope sind kategoriale Variablen
X = sm.add_constant(X)
model = smf.glm("C(status)~age+sex+C(cp)+trestbps+chol+fbs+C(restecg)+thalach+exang+old
                data=X_train, family=sm.families.Binomial()).fit()

# Vorhersage mit Testdaten
pred = model.predict(X_test)
pred_class = ["gesund" if x >0.5 else "krank" for x in pred]

# Modellreport
print(classification_report(pred_class, y_test))
```

```
              precision    recall  f1-score   support

      gesund       0.85      0.76      0.80        45
       krank       0.69      0.81      0.75        31

    accuracy                           0.78        76
   macro avg       0.77      0.78      0.77        76
weighted avg       0.79      0.78      0.78        76
```

```
C:\Users\dea40349\Anaconda3\lib\site-packages\statsmodels\tsa\tsatools.py:142: FutureWarning: In a future vers
of concat except for the argument 'objs' will be keyword-only
  x = pd.concat(x[::order], 1)
```

In [26]:

```
model.summary()
```

Generalized Linear Model Regression Results

Dep. Variable:	['C(status)[gesund]', 'C(status)[krank]']	**No. Observations:**	227
Model:	GLM	**Df Residuals:**	204
Model Family:	Binomial	**Df Model:**	22
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-64.088
Date:	Fri, 31 Dec 2021	**Deviance:**	128.18
Time:	10:43:24	**Pearson chi2:**	296.
No. Iterations:	7		
Covariance Type:	nonrobust		

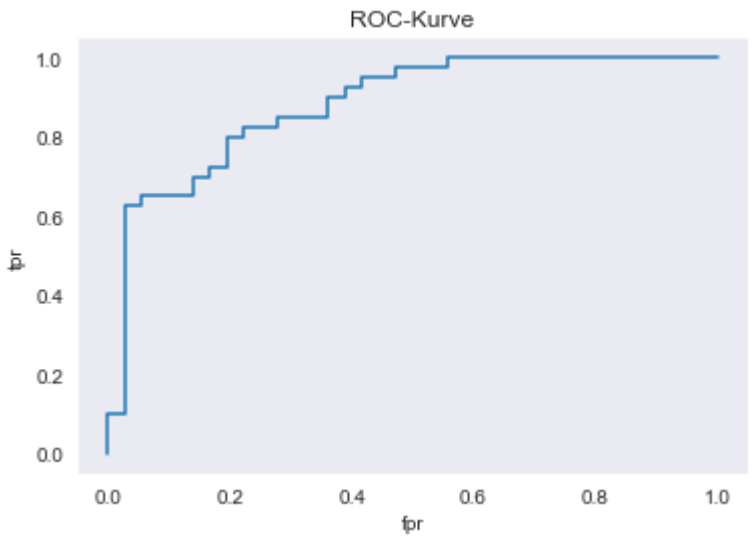
	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	9.7744	3.921	2.493	0.013	2.089	17.460
C(cp)[T.2.0]	-1.6902	1.002	-1.687	0.092	-3.654	0.274
C(cp)[T.3.0]	-0.3421	0.888	-0.385	0.700	-2.082	1.398
C(cp)[T.4.0]	-2.4047	0.894	-2.688	0.007	-4.158	-0.652
C(restecg)[T.1.0]	-0.4427	3.182	-0.139	0.889	-6.679	5.793
C(restecg)[T.2.0]	-0.2381	0.485	-0.491	0.623	-1.188	0.712
C(slope)[T.2.0]	-1.6602	0.561	-2.960	0.003	-2.760	-0.561
C(slope)[T.3.0]	-1.1133	1.079	-1.032	0.302	-3.229	1.002
ca[T.1.0]	-1.8893	0.658	-2.871	0.004	-3.179	-0.599
ca[T.2.0]	-3.2605	0.927	-3.517	0.000	-5.078	-1.443
ca[T.3.0]	-2.6116	1.245	-2.098	0.036	-5.051	-0.172
ca[T.?]	-1.2902	4.095	-0.315	0.753	-9.316	6.735
thal[T.6.0]	-0.0377	1.053	-0.036	0.971	-2.101	2.026
thal[T.7.0]	-2.1106	0.589	-3.584	0.000	-3.265	-0.956
thal[T.?]	-2.9045	2.331	-1.246	0.213	-7.472	1.663
age	0.0217	0.031	0.689	0.491	-0.040	0.083
sex	-0.8928	0.681	-1.312	0.190	-2.227	0.441
trestbps	-0.0419	0.015	-2.745	0.006	-0.072	-0.012
chol	-0.0095	0.006	-1.658	0.097	-0.021	0.002
fbs	0.8828	0.758	1.165	0.244	-0.602	2.368
thalach	0.0174	0.016	1.104	0.270	-0.014	0.048
exang	-0.9981	0.567	-1.760	0.078	-2.110	0.113
oldpeak	-0.3761	0.290	-1.296	0.195	-0.945	0.193

Erstellen wir noch eine ROC-Kurve und lassen die AUC berechnen:

In [27]:

```
from sklearn.metrics import roc_curve


y_test = [1 if x=="gesund" else 0 for x in y_test]
fpr, tpr, thresholds = roc_curve(y_test, pred)


plt.plot(fpr, tpr)
plt.title("ROC-Kurve")
plt.xlabel("fpr")
plt.ylabel("tpr")
plt.plot()
```

```
[]
```



In [28]:

```
from sklearn.metrics import roc_auc_score
pred_class = [1 if x=="gesund" else 0 for x in pred_class]
print("AUC = ", roc_auc_score(y_test, pred_class))
```

```
AUC =  0.7722222222222223
```