In [1]:

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

In [2]:

```
df = pd.read_excel("./age_car.xlsx")
df.head()
```

Out[2]:

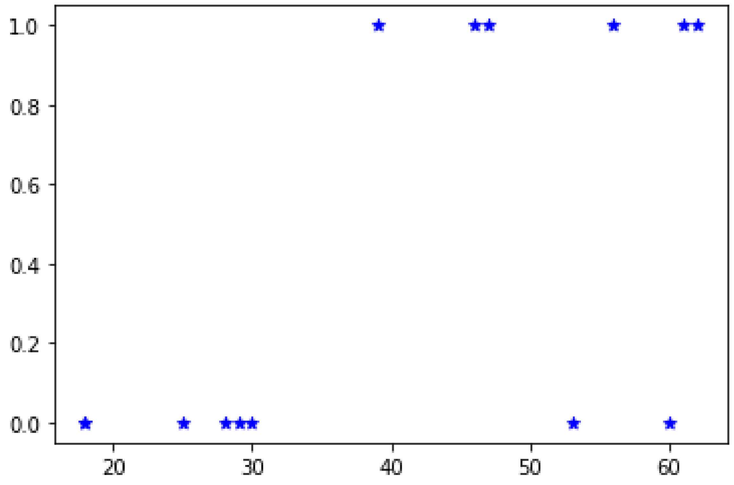
|  | age | have_car |
|---|---|---|
| **0** | 18 | 0 |
| **1** | 25 | 0 |
| **2** | 47 | 1 |
| **3** | 53 | 0 |
| **4** | 46 | 1 |

In [3]:

```
plt.scatter( np.array(df.age), np.array(df.have_car), marker='*',color='blue')
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0x2483bf07cd0>
```



In [4]:

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
x = np.array(df.age).reshape((14,-1))
print(x.shape)
y = np.array(df.have_car)
print(y.shape)
```

```
(14, 1)
(14,)
```

In [5]:

```
model = model.fit(x, y)
```

In [6]:

```
r_sq = model.score(x, y)
print(f"coefficient of determination: {r_sq}")
```

```
coefficient of determination: 0.37385129049508825
```

In [ ]:

```

```

''' When you’re applying .score(), the arguments are also the predictor x and response y, and the return value is $R^2$.

The attributes of model are .intercept_, which represents the coefficient $b_0$, and .coef_, which represents $b_1$: '''

In [7]:

```
print(f"intercept: {model.intercept_}")

print(f"slope: {model.coef_}")
```

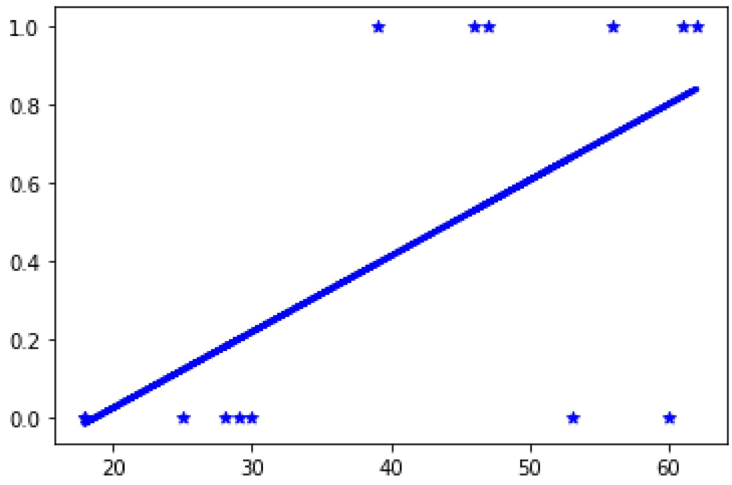
```
intercept: -0.366630076838863896
slope: [0.01946297]
```

In [8]:

```
# Make predictions using the testing set
y_pred = model.predict(x)
plt.scatter( np.array(df.age), np.array(df.have_car), marker='*',color='blue')
plt.plot(x, y_pred, color="blue", linewidth=3)
```

Out[8]:

```
[<matplotlib.lines.Line2D at 0x24837a62b50>]
```



In [ ]:

```

```

In [9]:

```
x_raw = np.array([0,0,0,0,0,0,18, 25, 47, 53, 46, 62, 61, 18, 29, 28, 30, 39, 100,100])
y_raw = np.array([0,0,0,0,0,0,0, 0, 1, 0, 1, 1, 0,  0, 0, 0, 0,1,1,1])
```

In [10]:

```
x_raw.shape,y_raw.shape
```

Out[10]:

```
((20,), (20,))
```

In [11]:

```
from scipy.optimize import curve_fit

def sigmoid(x, L ,x0, k, b):
    y = L / (1 + np.exp(-k*(x-x0))) + b
    return (y)

p0 = [max(y_raw), np.median(x_raw),1,min(y_raw)] # this is an mandatory initial guess

popt, pcov = curve_fit(sigmoid, x_raw, y_raw,p0, method='dogbox')
x_sigmoid = np.linspace(0, 100, 1000)
y_sigmoid = sigmoid(x_sigmoid, *popt)
```

In [ ]:

```

```

In [12]:

```
popt
```

Out[12]:

```
array([ 7.49999968e-01,  3.24388592e+01,  3.85232374e+00, -3.06981010e-06])
```

In [ ]:

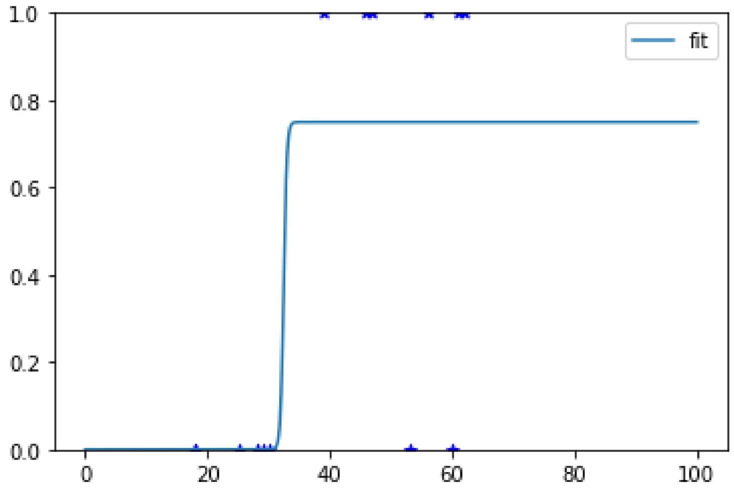
```

```

In [13]:

```
plt.scatter( np.array(x), np.array(y), marker='*',color='blue')
plt.plot(x_sigmoid,y_sigmoid, label='fit')
plt.ylim(0, 1)
plt.legend(loc='best')
```

Out[13]:

```
<matplotlib.legend.Legend at 0x24837a86d90>
```



In [ ]:

In [ ]:

In [ ]:

In [14]:

```
from sklearn.model_selection import train_test_split
```

In [ ]:

In [ ]:

In [15]:

```
X_train, X_test, y_train, y_test = train_test_split(
...     df[['age']],df.have_car, test_size=0.1, random_state=42)
```

In [16]:

```
X_train, X_test, y_train, y_test
```

Out[16]:

```
(    age
 0    18
 12   30
 5    56
 8    61
 2    47
 1    25
 13   39
 4    46
 7    62
 10   29
 3    53
 6    60,
     age
 9    18
 11   28,
 0     0
 12    0
 5     1
 8     1
 2     1
 1     0
 13    1
 4     1
 7     1
 10    0
 3     0
 6     0
 Name: have_car, dtype: int64,
 9     0
 11    0
 Name: have_car, dtype: int64)
```

In [17]:

```
from sklearn.linear_model import LogisticRegression
```

In [18]:

```
model = LogisticRegression()
model.fit(X_train, y_train)
```

Out[18]:

```
LogisticRegression()
```

In [19]:

```
model.predict(X_test)
```

Out[19]:

```
array([0, 0], dtype=int64)
```

In [20]:

```
X_test, y_test
```

Out[20]:

```
(    age
 9    18
 11   28,
 9     0
 11    0
 Name: have_car, dtype: int64)
```

In [21]:

```
model.score(X_test, y_test)
```

Out[21]:

```
1.0
```

In [22]:

```
model.predict_proba(X_test)
```

Out[22]:

```
array([[0.9173159 , 0.0826841 ],
       [0.81678517, 0.18321483]])
```

In [ ]:

```

```

In [23]:

```
y_test
```

Out[23]:

```
9     0
11    0
Name: have_car, dtype: int64
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]: