# Linear Regression with Single Variable

In [25]:

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

In [2]:

```
df =  pd.read_csv('homeprice.csv')
df
```

Out[2]:

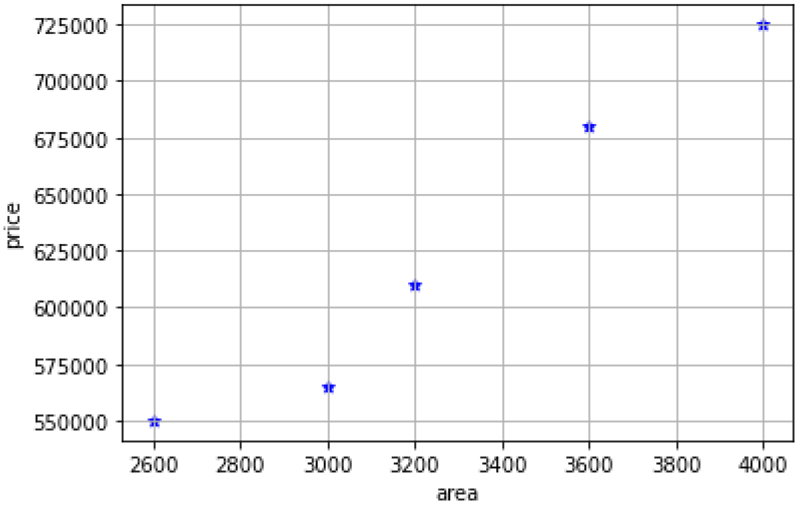
| | area | price |
|---|---|---|
| **0** | 2600 | 550000 |
| **1** | 3000 | 565000 |
| **2** | 3200 | 610000 |
| **3** | 3600 | 680000 |
| **4** | 4000 | 725000 |

In [3]:

```
%matplotlib inline
plt.xlabel('area')
plt.ylabel('price')
plt.grid()
plt.scatter(df.area, df.price, color = 'blue', marker = '*')
```

Out[3]:

```
<matplotlib.collections.PathCollection at 0x1e8115b23a0>
```



In [6]:

```
new_df =  df.drop('price', axis = "columns")
new_df
```

Out[6]:

| | area |
|---|---|
| **0** | 2600 |
| **1** | 3000 |
| **2** | 3200 |

```
     area
3    3600
4    4000
```

In [26]:
```
price = df.price
print(type(price))
np.array(price)
```

```
<class 'pandas.core.series.Series'>
```

Out[26]:
```
array([550000, 565000, 610000, 680000, 725000], dtype=int64)
```

In [10]:
```
#create linear regression object
reg = linear_model.LinearRegression()
reg.fit(new_df, price)
```

Out[10]:
```
LinearRegression()
```

### (1) Predict price of a home with area = 4500 sft

In [11]:
```
reg.predict([[4500]])
```

Out[11]:
```
array([791660.95890411])
```

In [12]:
```
reg.coef_ #value of m slope
```

Out[12]:
```
array([135.78767123])
```

In [13]:
```
reg.intercept_ #value of intercept c
```

Out[13]:
```
180616.43835616432
```

# Generate CSV file with list of home price predictions

In [39]:
```
area_df = pd.read_csv('area.csv')
area_df.head()
```

Out[39]:
```
     area
0    1000
1    1500
2    2300
3    3540
4    4120
```

In [40]:
```
p = reg.predict(area_df)
p
```

```
Out[40]: array([ 316404.10958904,  384297.94520548,  492928.08219178,
                 661304.79452055,  740061.64383562,  799808.21917808,
                 926090.75342466,  650441.78082192,  825607.87671233,
                 492928.08219178, 1402705.47945205, 1348390.4109589 ,
                1144708.90410959])
```

```
In [41]: area_df['predicted_prices'] = p
         area_df
```

Out[41]:

|  | area | predicted_prices |
|---|---|---|
| **0** | 1000 | 3.164041e+05 |
| **1** | 1500 | 3.842979e+05 |
| **2** | 2300 | 4.929281e+05 |
| **3** | 3540 | 6.613048e+05 |
| **4** | 4120 | 7.400616e+05 |
| **5** | 4560 | 7.998082e+05 |
| **6** | 5490 | 9.260908e+05 |
| **7** | 3460 | 6.504418e+05 |
| **8** | 4750 | 8.256079e+05 |
| **9** | 2300 | 4.929281e+05 |
| **10** | 9000 | 1.402705e+06 |
| **11** | 8600 | 1.348390e+06 |
| **12** | 7100 | 1.144709e+06 |

```
In [33]: area_df.to_csv('prediction.csv')
```

```
In [34]: from sklearn.metrics import accuracy_score
```

```
In [35]: accuracy_score(p, price)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-35-25507e757be4> in <module>
----> 1 accuracy_score(p, price)

G:\AnacondaInstallation\lib\site-packages\sklearn\utils\validation.py in inner_f(*args,
 **kwargs)
     70                           FutureWarning)
     71         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 72         return f(**kwargs)
     73     return inner_f
     74

G:\AnacondaInstallation\lib\site-packages\sklearn\metrics\_classification.py in accuracy
_score(y_true, y_pred, normalize, sample_weight)
    185
    186     # Compute accuracy for each possible representation
--> 187     y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    188     check_consistent_length(y_true, y_pred, sample_weight)
    189     if y_type.startswith('multilabel'):
```

```
G:\AnacondaInstallation\lib\site-packages\sklearn\metrics\_classification.py in _check_t
argets(y_true, y_pred)
     79     y_pred : array or indicator matrix
     80     """
---> 81     check_consistent_length(y_true, y_pred)
     82     type_true = type_of_target(y_true)
     83     type_pred = type_of_target(y_pred)

G:\AnacondaInstallation\lib\site-packages\sklearn\utils\validation.py in check_consisten
t_length(*arrays)
    253     uniques = np.unique(lengths)
    254     if len(uniques) > 1:
--> 255         raise ValueError("Found input variables with inconsistent numbers of"
    256                          " samples: %r" % [int(l) for l in lengths])
    257 

ValueError: Found input variables with inconsistent numbers of samples: [13, 5]
```

```
In [ ]:
```