Lab 8: Regression with Scikit-Learn

The objective of this notebook is to learn about the **Scikit-Learn** library (official documentation) and **regression**.

In this lab, we will train a regression model that predicts the price of the house given some input features such as 'price', 'area', 'bedrooms', 'bathrooms', 'stories', 'parking'.

Outline

First, run the following cell to import some useful libraries to complete this Lab. If not already done, you must install them in your virtual environment

In [31]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn import svm

from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor
```

1. Load dataset

Firstly, you will load the first dataset for this lab into a DataFrame `df`. The dataset is stored in the csv file from the following path `"data_lab8/Housing.csv"`.

In [3]:
```
data_path = "data_lab8/Housing.csv"
df = pd.read_csv(data_path)
```

In [4]:
```
df.head()
```

Out[4]:

	price	area	bedrooms	bathrooms	stories	mainroad	guestroom	basement	hotwa
0	13300000	7420	4	2	3	yes	no	no	
1	12250000	8960	4	4	4	yes	no	no	
2	12250000	9960	3	2	2	yes	no	yes	
3	12215000	7500	4	2	2	yes	no	yes	
4	11410000	7420	4	1	2	yes	yes	yes	

In [5]:
```
print(f"There are {len(df)} samples in the dataset.")
```

```
There are 545 samples in the dataset.
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'mainroad',
       'guestroom', 'basement', 'hotwaterheating', 'airconditioning',
       'parking', 'prefarea', 'furnishingstatus'],
      dtype='object')
```

As you can see, the dataset is composed of many columns. Some are **numerical attributes** (i.e., `price`, `area`, `bedrooms`, `bathrooms`, `stories`, and `parking`). In contrast, other columns are categorical attributes (i.e., `mainroad`, `guestroom`, `basement`, `hotwaterheating`, `airconditioning`, `prefarea`, and `furnishingstatus`). Remember that Machine Learning algorithms works only with numerical features. Therefore, categorical feature must be encoded to numbers as a pre-prcessing step. We will learn more about pre-processing in the next lectures. For now, let's focus on numerical features.

Exercise 1.1

Select the list of columns in `numerical_columns` from the DataFrame `df` and assign the selected subset DataFrame to the same variable `df`.

In [7]:
```
numerical_columns = ['price', 'area', 'bedrooms', 'bathrooms', 'stories', 'p

#### START CODE HERE ####
#### Approximately 1 line ####
df = df[numerical_columns]

#### END CODE HERE ####
```

In [8]:
```
df.head()
```

Out[8]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

Expected output

price	area	bedrooms	bathrooms	stories	parking	
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

2. Data Exploration

Exercise 2.1

Let's start by exploring the **target column** `price`. Compute the **mean**, the **standard deviation**, and the **variance** of the `price` column. Store the **mean**, the **standard deviation**, and the **variance** in the variables `price_mean`, `price_std`, and `price_var` respectively.

In [13]:
```
#### START CODE HERE ####
#### Approximately 2 line ####
price_mean = df["price"].mean()
price_std = df["price"].std()
price_var = df["price"].var()
#### END CODE HERE ####
```

In [14]:
```
print(f"Price mean: {price_mean:.2f}")
print(f"Price standard deviation: {price_std:.2f}")
print(f"Price variance: {price_var:.2f}")
```

```
Price mean: 4766729.25
Price standard deviation: 1870439.62
Price variance: 3498544355820.57
```

Expected output

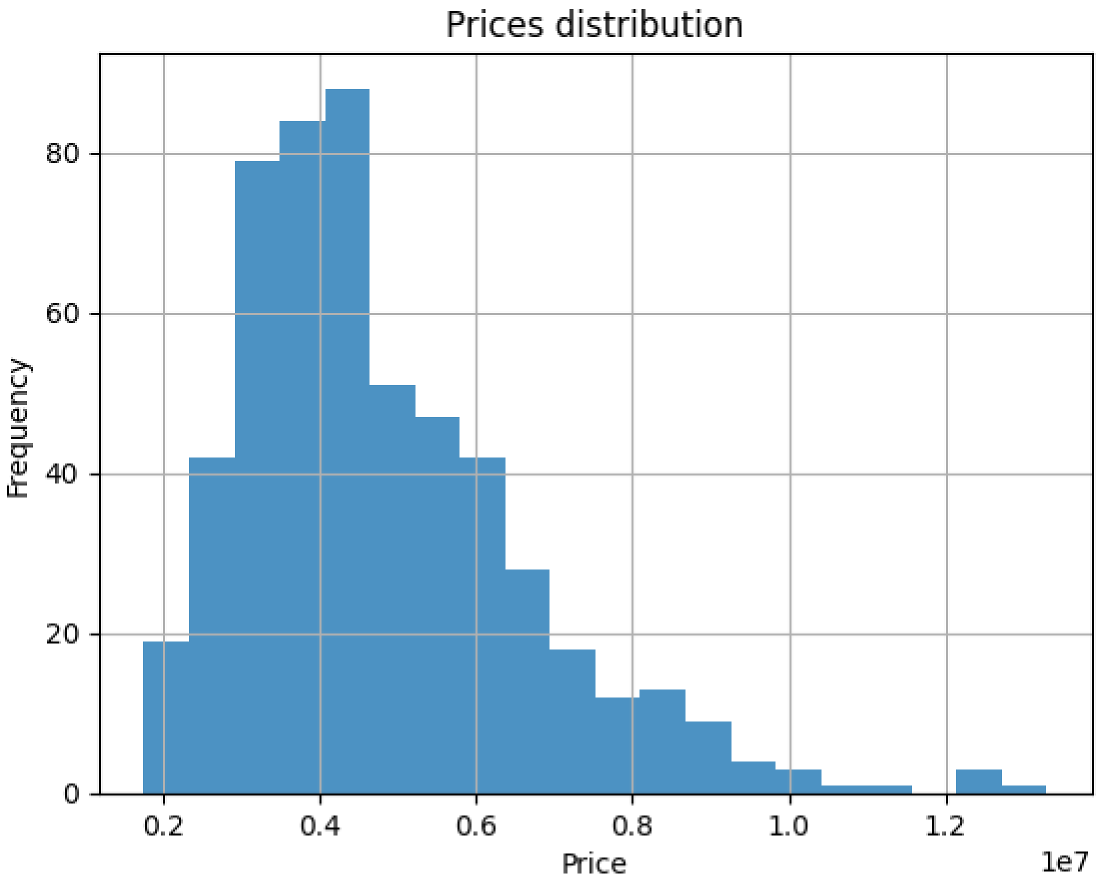
`Price mean: 4766729.25`

`Price standard deviation: 1870439.62`

`Price variance: 3498544355820.57`

The next cell plots the distributions of the prices. Please run the following cell to show the plot.

In [11]:
```
ax = df["price"].plot.hist(bins=20, alpha=0.8)
ax.set_xlabel("Price")
ax.set_title("Prices distribution")
ax.grid(True)
plt.show()
```

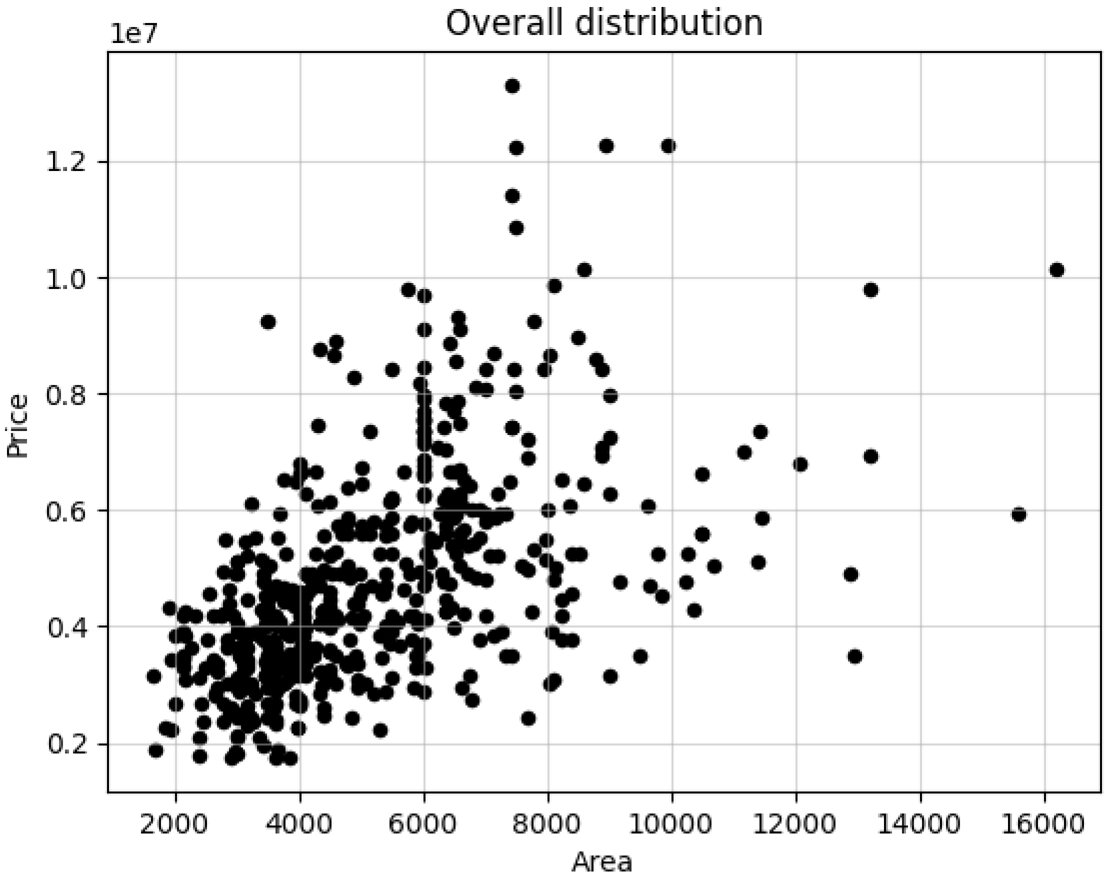


3. Linear Regression with 1D input features

Now you will implement a **Linear Regression** using a one-dimensional input feature (i.e., the *area* of the houses). Therefore, the task is to predict the *Price* of the houses given the *Area*.

Firstly, run the next cell to plot the points in the space.

In [15]:
```
df.plot(x='area', y='price', kind='scatter', c='black')
plt.title("Overall distribution")
plt.xlabel("Area")
plt.ylabel("Price")
plt.grid(True, alpha=0.5)
plt.show()
```



You can see that some **noisy points** are present. **Noisy points** can affect the performance of your learning algorithms. Indeed, some points have a really big area far from the distribution of the other points. We will perform a simple pre-processing step to remove the points with *area* >= 12000.

Run the next cell to perform the pre-processing.

In [16]:
```
df_1d = df.loc[df.area < 12000]
```

In [17]:
```
print(len(df_1d))
```

```
538
```

Now, we will select only the *Area* as input feature `df_X_1d` and the *Price* as target variable `df_Y_1d`.

In [18]:
```
df_X_1d = df_1d[["area"]]
df_Y_1d = df_1d[["price"]]
```

As usual, we will split our data into training and test set.

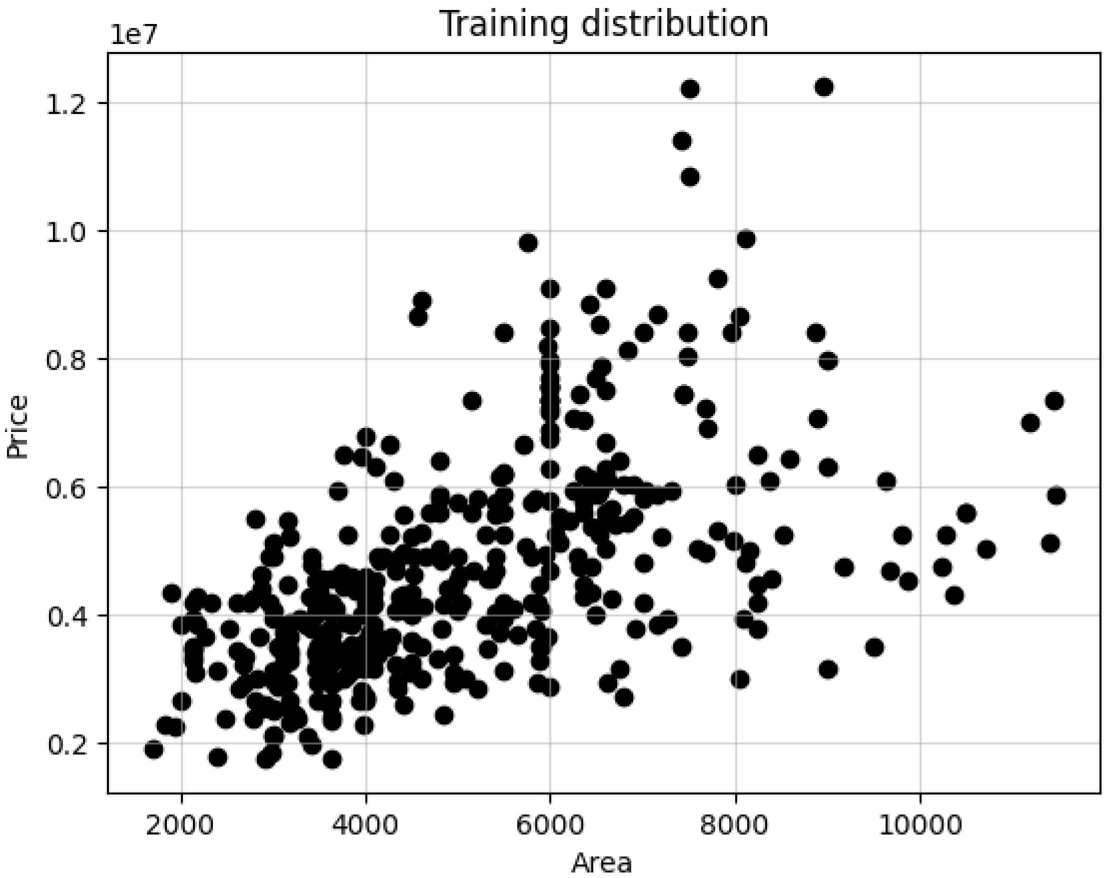
In [22]:
```
X_train_1d, X_test_1d, y_train_1d, y_test_1d = train_test_split(df_X_1d, df_
```

The following cell plots the distribution of the training points in the plane. Run the next cell to visualize the training points.

In [23]:
```
fig, ax = plt.subplots()
ax.scatter(x=X_train_1d, y=y_train_1d, c='black')

ax.set_title("Training distribution")
```

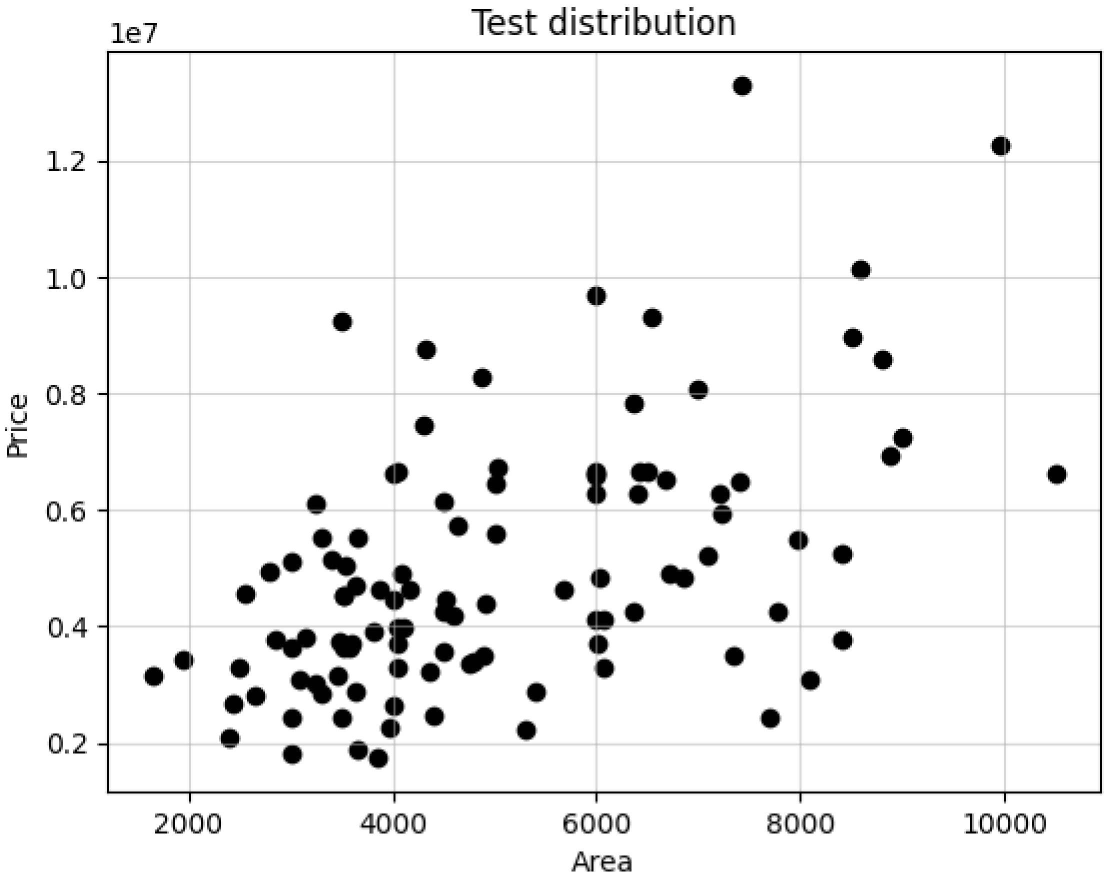
```
ax.set_xlabel("Area")
ax.set_ylabel("Price")
plt.grid(True, alpha=0.5)
plt.show()
```



The following cell plots the distribution of the test points in the plane. Run the next cell to visualize the test points.

In [24]:
```
fig, ax = plt.subplots()
ax.scatter(x=X_test_1d, y=y_test_1d ,c='black')

ax.set_title("Test distribution")
ax.set_xlabel("Area")
ax.set_ylabel("Price")
plt.grid(True, alpha=0.5)
plt.show()
```



Exercise 3.1

Create a `LinearRegression()` object and fit the linear regression on the training data. Replace `None` with your code.

In [25]:
```
#### START CODE HERE ####
#### Approximately 2 line ####
regr = linear_model.LinearRegression()
regr.fit(X_train_1d, y_train_1d)
#### END CODE HERE ####
```

Out[25]:
```
▾ LinearRegression
LinearRegression()
```

Exercise 3.2

Predict the prices of the houses for your **training data** in a variable `y_pred_train_1d`.

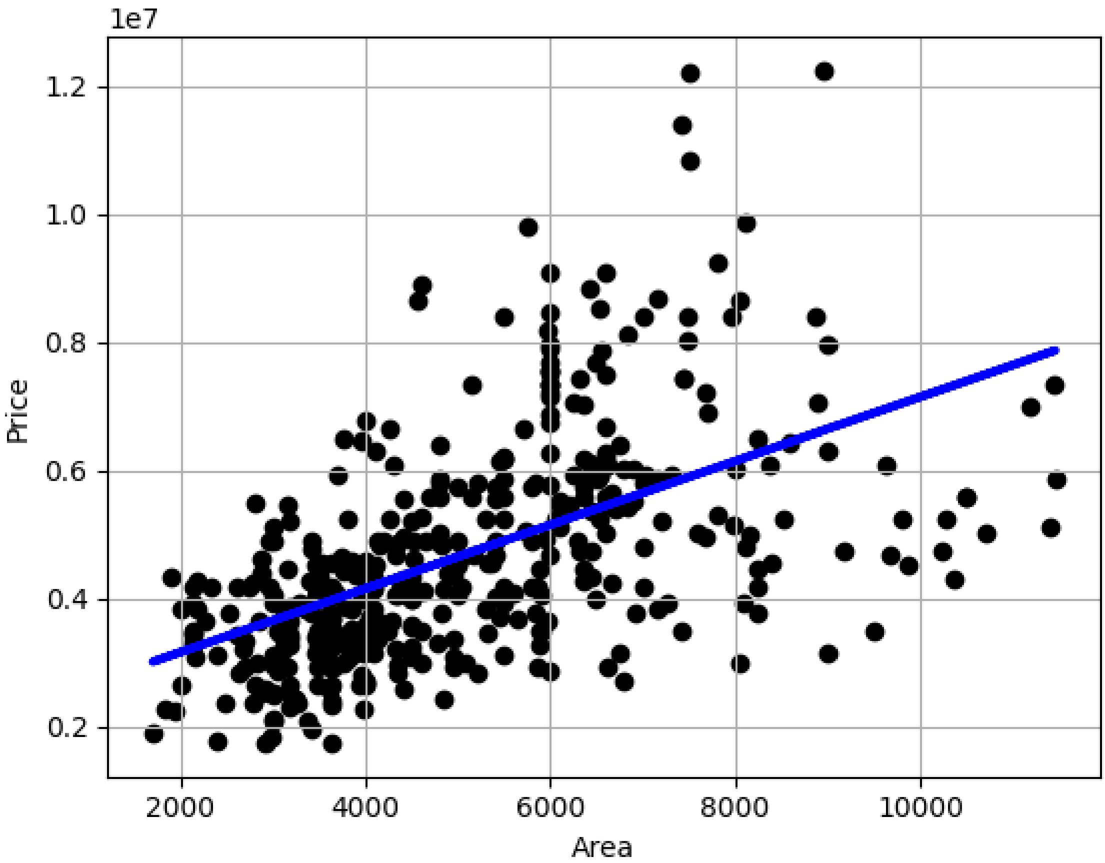
In [26]:
```
#### START CODE HERE ####
#### Approximately 1 line ####
y_pred_train_1d = regr.predict(X_train_1d)
#### END CODE HERE ####
```

The next cell visualize the learned straight line on your **training data**. Run the following cell to visualize the learned line.

In [27]:
```
plt.scatter(X_train_1d, y_train_1d, color="black")
plt.plot(X_train_1d, y_pred_train_1d, color="blue", linewidth=3)

plt.xlabel("Area")
plt.ylabel("Price")
plt.xticks()
plt.yticks()

plt.grid(True)
plt.show()
```



Exercise 3.3

Predict the prices of the houses for your **test data** in a variable `y_pred_test_1d`.

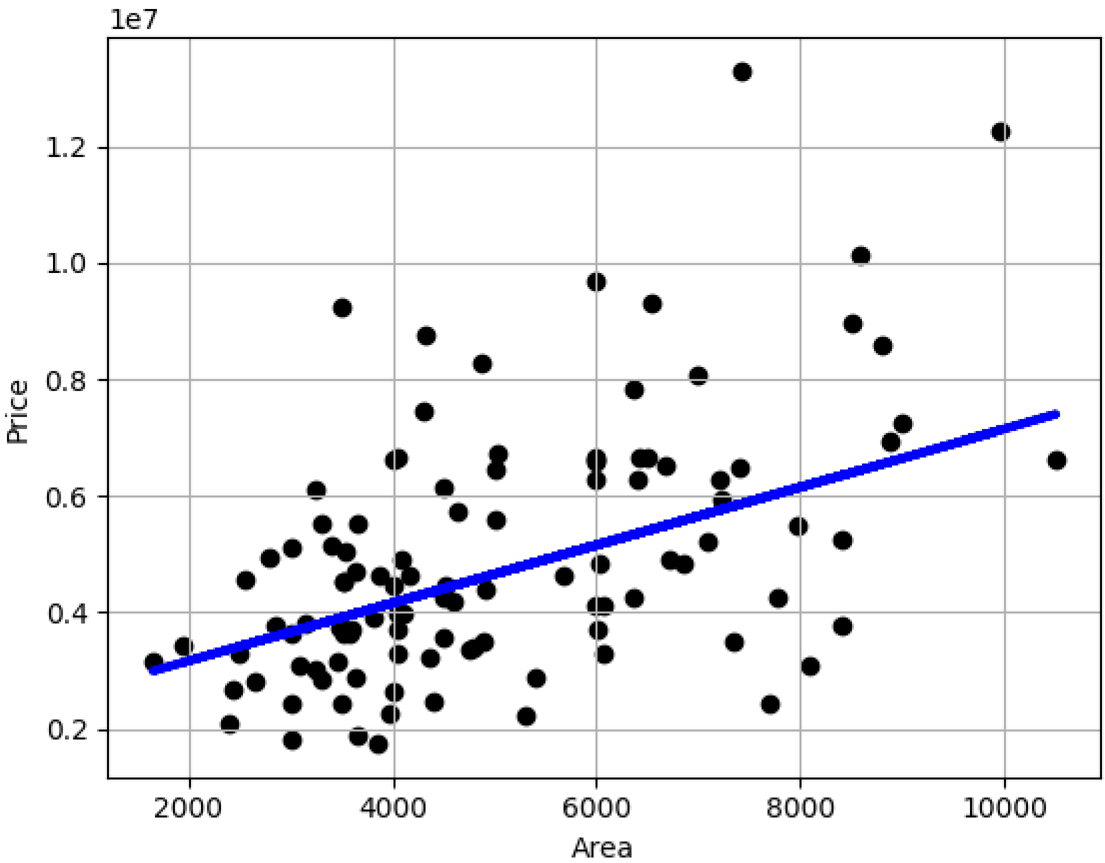
In [28]:
```
#### START CODE HERE ####
#### Approximately 1 line ####
y_pred_test_1d = regr.predict(X_test_1d)
#### END CODE HERE ####
```

The next cell visualize the learned straight line on your training data and the points of the **test data**. Run the following cell to visualize the learned line.

In [29]:
```
plt.scatter(X_test_1d, y_test_1d, color="black")
plt.plot(X_test_1d, y_pred_test_1d, color="blue", linewidth=3)

plt.xlabel("Area")
plt.ylabel("Price")
plt.xticks()
plt.yticks()
```

```
plt.grid(True)
plt.show()
```



Exercise 3.4

Compute the **Mean Absolute Error**, the **Mean Squared Error** and the **R2** in the variables `mae_test`, `mse_test`, and `r2_test`, respectively. Replace `None` with your code.

You read more on such metrics in the official documentation:

- mean_absolute_error
- mean_squared_error
- r2_score

In [33]:
```
#### START CODE HERE ####
#### Approximately 3 line ####
mae_test = mean_absolute_error(y_test_1d, y_pred_test_1d)
mse_test = mean_squared_error(y_test_1d, y_pred_test_1d)
r2_test = r2_score(y_test_1d, y_pred_test_1d)
#### END CODE HERE ####
```

In [38]:
```
print(f"Mean of the prices: {price_mean}")
print(f"Std of the prices: {price_std}")
print(f"Variance of the prices: {price_var}")

print(f"\nMean Absolute Error on test data: {mae_test}")
print(f"Mean Squared Error on test data: {mse_test}")
print(f"R2 score on test data: {r2_test}")
```

```
Mean of the prices: 4766729.247706422
Std of the prices: 1870439.6156573922
Variance of the prices: 3498544355820.573

Mean Absolute Error on test data: 1407509.8013314893
Mean Squared Error on test data: 3575212178202.1855
R2 score on test data: 0.2467447918823492
```

The model does not seem to perform very well. Let's see if we can improve by using all the input features.

4. Regression with all input features

Now you will train and evaluate several regression models with **all the numerical input features**.

In [39]:
```
df.head()
```

Out[39]:

	price	area	bedrooms	bathrooms	stories	parking
0	13300000	7420	4	2	3	2
1	12250000	8960	4	4	4	3
2	12250000	9960	3	2	2	2
3	12215000	7500	4	2	2	3
4	11410000	7420	4	1	2	2

Run the next cell to select all the numerical input features as input.

In [40]:
```
df_X = df.loc[:, "area":]
df_y = df[["price"]]
```

Run the next cell to split the data into training and test sets.

In [42]:
```
X_train, X_test, y_train, y_test = train_test_split(df_X, df_y, test_size=0.
```

In [43]:
```
X_train.head()
```

Out[43]:

	area	bedrooms	bathrooms	stories	parking
46	6000	3	2	4	1
93	7200	3	2	1	3
335	3816	2	1	1	2
412	2610	3	1	2	0
471	3750	3	1	2	0

In [44]:
```
X_test.head()
```

Out[44]:

	area	bedrooms	bathrooms	stories	parking
316	5900	4	2	2	1
77	6500	3	2	3	0
360	4040	2	1	1	0
90	5000	3	1	2	0
493	3960	3	1	1	0

As you can see, the input features have very **different scales**. As discussed in previous labs, features with different scales differentially impact the calculation of validation metrics. We must therefore perform, separately for each column, the **normalization of the input features**.

However, this time we have both training and test set. When you have both training and test, you have to **calculate statistics for the normalization on the training** (i.e., with the `fit_transform()` method) and **use those statistics on the test set** (i.e., with the `transform()` method). This is because the model cannot learn on the test data. This data simulates data never seen by the model on which it will have to make predictions. Therefore, they cannot even be used to estimate some statistics about the data.

In this case, we want to perform **min-max normalization** of the dataset. To achieve this in scikit-learn is simple. There is a function in the pre-processing module to do this. However, as introduced before, the min and max are calculated only on the training and are used to normalize both the training and the test.

If this step is not clear to you, don't worry. We will see it in detail when we talk about data pre-processing.

Run the following cell to perform the Min-Max normalization.

In [45]:
```
from sklearn import preprocessing

min_max_scaler = preprocessing.MinMaxScaler()

X_train_processed = min_max_scaler.fit_transform(X_train)
X_train_processed = pd.DataFrame(X_train_processed, columns=X_train.columns)

X_test_processed = min_max_scaler.transform(X_test)
X_test_processed = pd.DataFrame(X_test_processed, columns=X_test.columns)
```

In [46]:
```
X_train_processed.head()
```

Out[46]:

	area	bedrooms	bathrooms	stories	parking
0	0.298969	0.4	0.333333	1.000000	0.333333
1	0.381443	0.4	0.333333	0.000000	1.000000
2	0.148866	0.2	0.000000	0.000000	0.666667
3	0.065979	0.4	0.000000	0.333333	0.000000
4	0.144330	0.4	0.000000	0.333333	0.000000

In [47]:
```
X_test_processed.head()
```

Out[47]:

	area	bedrooms	bathrooms	stories	parking
0	0.292096	0.6	0.333333	0.333333	0.333333
1	0.333333	0.4	0.333333	0.666667	0.000000
2	0.164261	0.2	0.000000	0.000000	0.000000
3	0.230241	0.4	0.000000	0.333333	0.000000
4	0.158763	0.4	0.000000	0.000000	0.000000

As you can see, after normalization, all features in the training set are in the range [0, 1].

Exercise 4.1

Now you will train and evalaute several regression models on the preprocessed data. Note that you should use `X_train_processed` and `X_test_processed` as input of your models.

This exercise is open. So it's up to you to choose regression models from those available on scikit-learn, train and validate them.

In [ ]:
```
```