Explanation:

pandas as pd Used for data manipulation and analysis. It provides data structures like DataFrame and functions for cleaning and transforming data.

numpy as np Fundamental package for numerical computing in Python. It provides support for arrays, mathematical functions, and linear algebra.

matplotlib.pyplot as plt Used for data visualization. It offers functions to create line plots, scatter plots, histograms, and more.

from sklearn import linear_model Imports Scikit-learn's linear modeling tools, including models like LinearRegression, LogisticRegression, and others. These are used for performing both regression and classification tasks.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

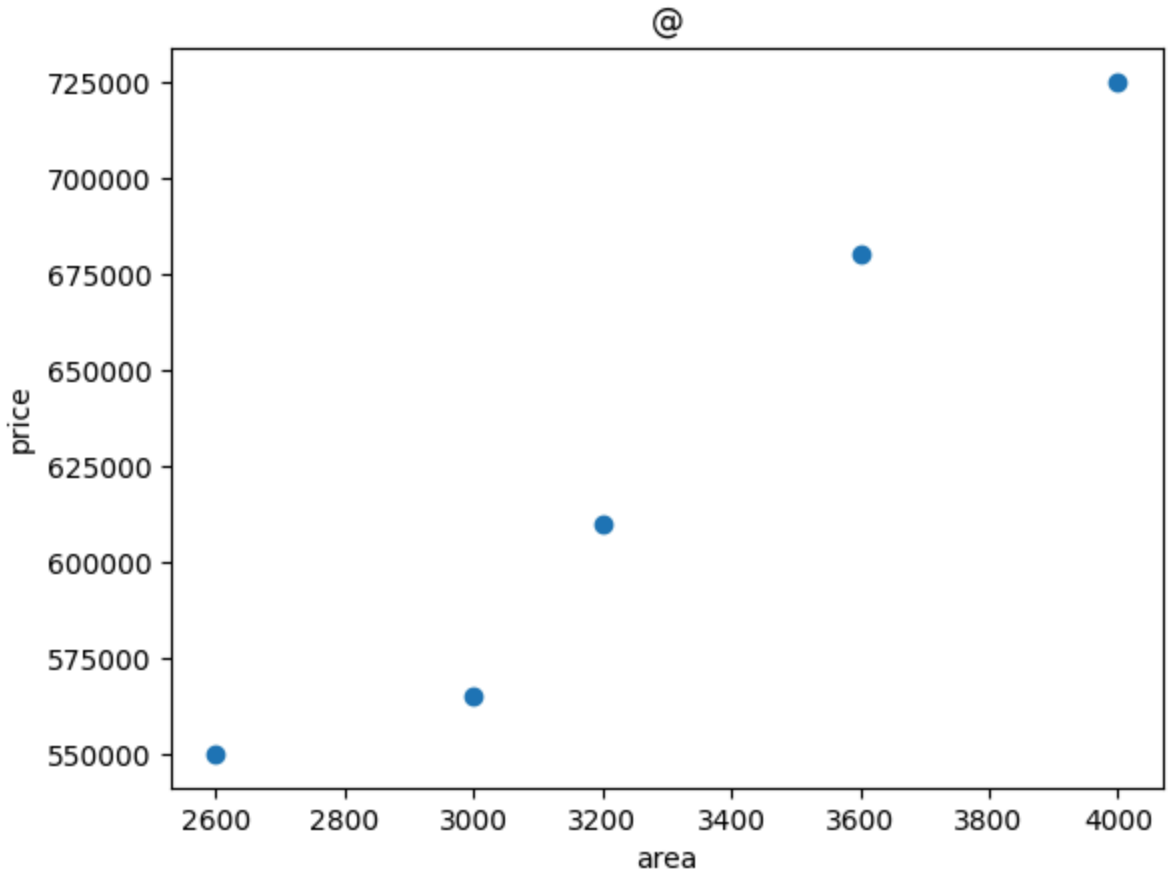
Creating a Simple Housing Dataset

In [2]:
```
data = pd.DataFrame({'area': [2600,3000,3200,3600,4000],
       'price': [550000, 565000,610000,680000,725000]
       })
data
```

Out[2]:

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

In [3]:
```
plt.scatter(data.area, data.price)
plt.xlabel('area')
plt.ylabel('price')
plt.title('@')
plt.show()
```



Training a Linear Regression Model.

LinearRegression(): Creates a linear regression model from Scikit-learn's linear_model module. This model finds the best-fit line that predicts price based on area.

In [4]:
```
reg = linear_model.LinearRegression()
reg.fit(data[['area']],data.price)
```

Out[4]:
```
▾ LinearRegression

LinearRegression()
```

This uses the trained linear regression model to predict the price of a house with an area of 3300 square feet.

In [5]:
```
predicted_price=reg.predict([[3300]])
print(f"Predicted price for 3300 sq ft: ${predicted_price[0]:,.2f}")
```

```
Predicted price for 3300 sq ft: $628,715.75
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

Getting the Slope (Coefficient) of the Linear Model. Explanation:

This returns the coefficient (also called the slope) of the trained linear regression model.

In this context, it represents the estimated change in price per additional square foot of house area.

For every additional 1 square foot of area, the predicted house price increases by approximately $135.79.

In [6]:
```
reg.coef_

print(f"Price increases by ${reg.coef_[0]:,.2f} per additional sq ft")
```

```
Price increases by $135.79 per additional sq ft
```

Explanation:

This returns the intercept of the trained linear regression model.

It represents the predicted price when the area is 0 square feet.

In the equation of a line:

In [7]:
```
reg.intercept_
```

Out[7]:
```
180616.43835616432
```

m=coef, b=intercept, mx+b

In [8]:
```
135.78767123*3300+180616.43835616432
```

Out[8]:
```
628715.7534151643
```

In [9]:
```
predicted_price = reg.predict([[5000]])
print(f"Predicted price for 5000 sq ft: ${predicted_price[0]:,.2f}")
```

```
Predicted price for 5000 sq ft: $859,554.79
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

Pandas Dataframe

In [10]:
```
d = pd.DataFrame({
    'area': [1000,1500,2300,3540,4120,4560,5490,3460,4750,2300,9000,8600,7100]
})
d
```

Out[10]:

	area
0	1000
1	1500
2	2300
3	3540
4	4120
5	4560
6	5490
7	3460
8	4750
9	2300
10	9000
11	8600
12	7100

Predicting House Prices for Multiple Areas

Explanation:

This uses the trained linear regression model (reg) to predict house prices for all the values in the d['area'] column.

The result, p, is a NumPy array of predicted prices that corresponds to each house size in d.

In [11]:
```
p = reg.predict(d)
```

Adding Predicted Prices to the DataFrame

Explanation:

This line adds a new column called 'prices' to the DataFrame d.

It assigns the predicted house prices (from the model) to each corresponding house area.

In [12]:
```
d['prices'] = p
```

This will convert the prices column to a string with two decimal places

x is the argument (each value in the prices column).

f"${x:,.2f}" is the expression that formats the value x as a string.

f"${x:,.2f}" Formatting: This part is using f-string formatting (formatted string literals), which is a way to embed expressions inside string literals. The format inside the curly braces {} defines how x should be displayed.

$: This adds the dollar sign at the start of the formatted string.

x: The value of x (the price) is what you are formatting.

:: The colon (:) starts the formatting specifiers.

,: The comma adds a thousands separator, which is commonly used in numbers to make them more readable (e.g., 1,000instead of1000`).

.2f:

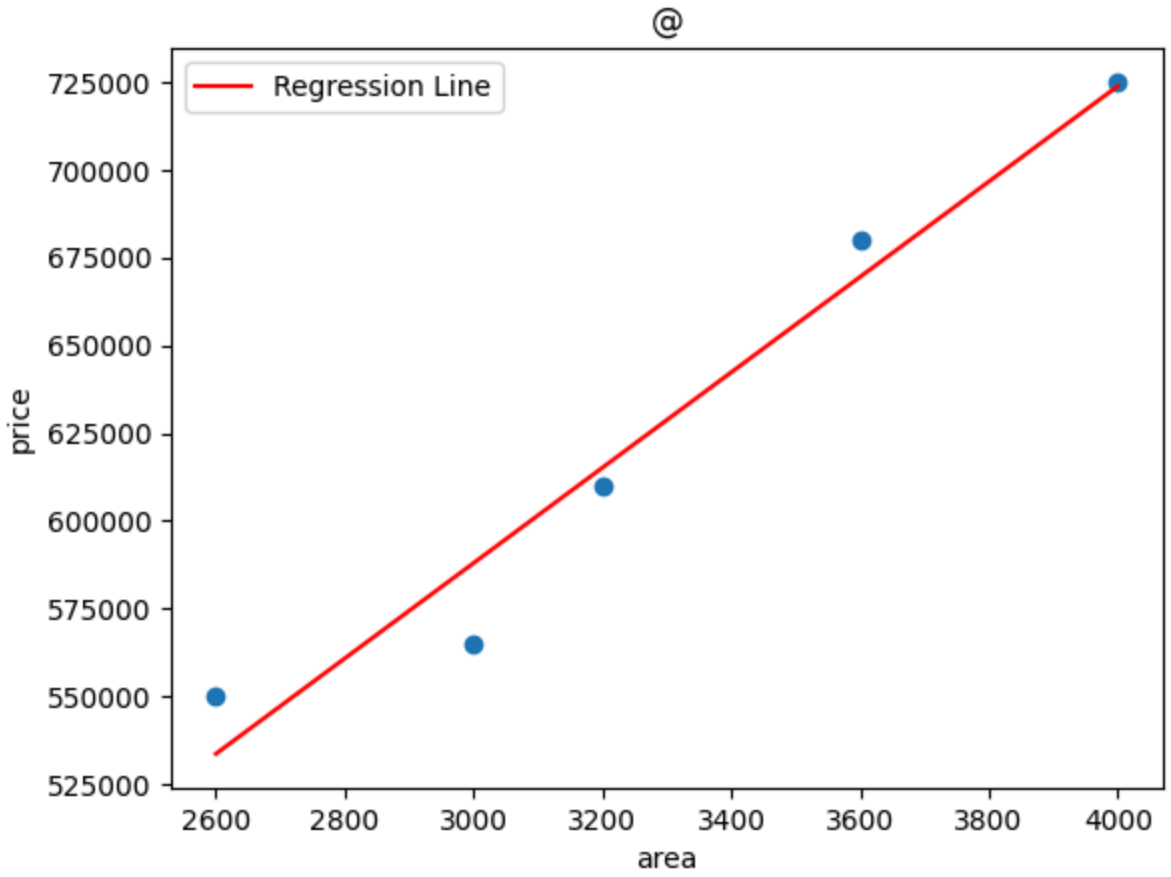
.2 specifies the number of decimal places you want to display. In this case, it's 2 decimal places (e.g., 316404.11).

f stands for floating-point number. It tells Python to format the number as a float, meaning you want to show decimal places.

In [13]:
```
d['prices'] = d['prices'].apply(lambda x: f"${x:,.2f}")
print(d)
```

```
    area         prices
0   1000    $316,404.11
1   1500    $384,297.95
2   2300    $492,928.08
3   3540    $661,304.79
4   4120    $740,061.64
5   4560    $799,808.22
6   5490    $926,090.75
7   3460    $650,441.78
8   4750    $825,607.88
9   2300    $492,928.08
10  9000  $1,402,705.48
11  8600  $1,348,390.41
12  7100  $1,144,708.90
```

In [14]:
```
plt.scatter(data.area, data.price)
plt.xlabel('area')
plt.ylabel('price')
plt.title('@')
plt.plot(data.area,reg.predict(data[['area']]),color='r', label='Regression Li
plt.legend()
plt.show()
```



In [15]:
```
income_canada = pd.DataFrame({
    'year': [1970,
1971,1972,1973,1974,1975,
1976,
1977,
1978,
1979,
1980,
1981,
1982,
1983,1984,
1985,

1986,
1987,
1988,
1989,
1990,
1991,
1992,
1993,
1994,
1995,
1996,
1997,
1998,
1999,
2000,
2001,
2002,
2003,
```

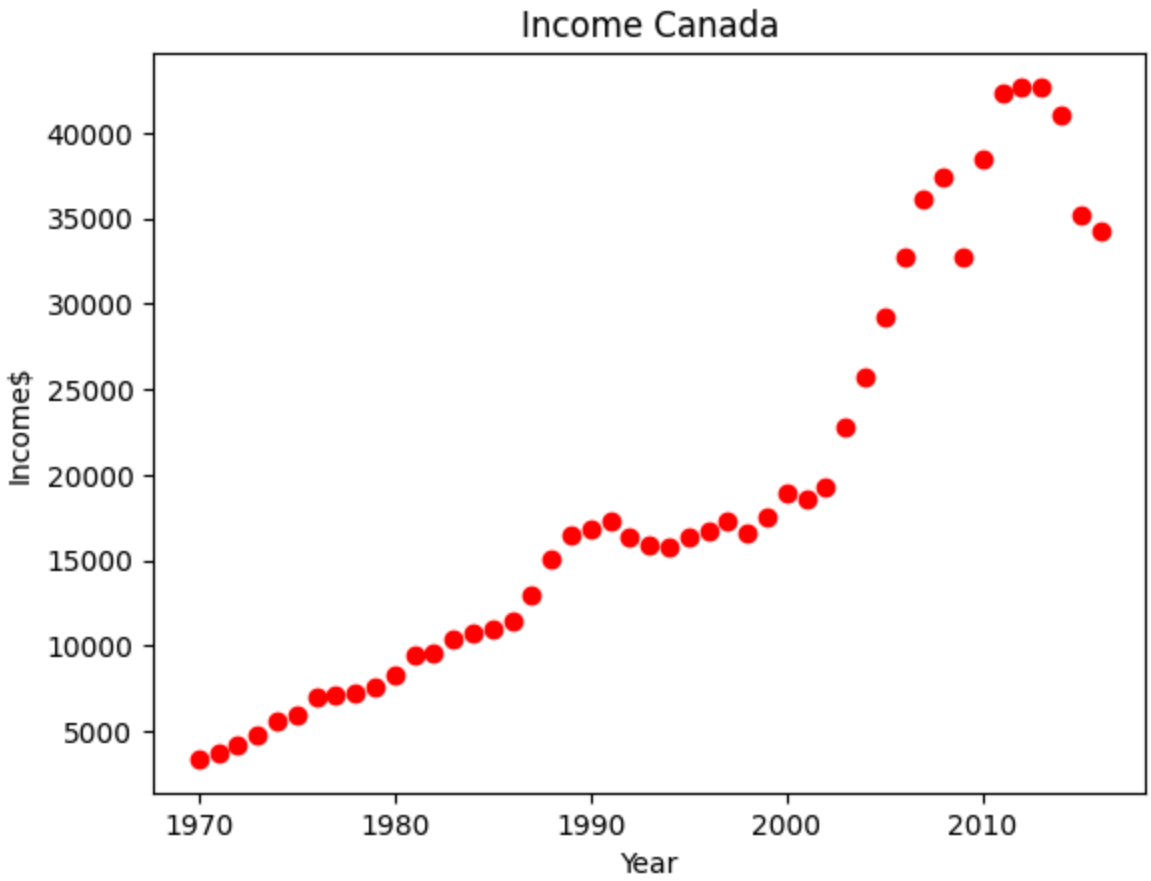
```
2004,
2005,
2006,
2007,
2008,
2009,
2010,
2011,
2012,
2013,
2014,
2015,
2016],
    'income':[
3399.299037,
3768.297935,
4251.175484,
4804.463248,
5576.514583,
5998.144346,
7062.131392,
7100.12617,
7247.967035,
7602.912681,
8355.96812,
9434.390652,
9619.438377,
10416.53659,
10790.32872,
11018.95585,
11482.89153,
12974.80662,
15080.28345,
16426.72548,
16838.6732,
17266.09769,
16412.08309,
15875.58673,
15755.82027,
16369.31725,
16699.82668,
17310.75775,
16622.67187,
17581.02414,
18987.38241,
18601.39724,
19232.17556,
22739.42628,
25719.14715,
29198.05569,
32738.2629,
36144.48122,
37446.48609,
32755.17682,
38420.52289,
42334.71121,
42665.25597,
42676.46837,
41039.8936,
35175.18898,
```

```
34229.19363]
})
income_canada.head(10)
```

Out[15]:

	year	income
0	1970	3399.299037
1	1971	3768.297935
2	1972	4251.175484
3	1973	4804.463248
4	1974	5576.514583
5	1975	5998.144346
6	1976	7062.131392
7	1977	7100.126170
8	1978	7247.967035
9	1979	7602.912681

In [16]:
```
plt.scatter(income_canada.year, income_canada.income, color = 'red')
plt.xlabel('Year')
plt.ylabel('Income$')
plt.title('Income Canada')
plt.show()
```



In [17]:
```
regression = linear_model.LinearRegression()
regression.fit(income_canada[['year']], income_canada.income)
```

Out[17]:
```
▾ LinearRegression
LinearRegression()
```

In [18]:
```
prediction_2020 = regression.predict([[2020]])

print(f"Predicted income for 2020: ${prediction_2020[0]:,.2f}")
```

```
Predicted income for 2020: $41,288.69
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

In [19]:
```
regression.coef_
```

Out[19]:
```
array([828.46507522])
```

In [20]:
```
regression.intercept_
```

Out[20]:
```
-1632210.7578554575
```

m*x+b,(m=coef, b=intercept)

In [21]:
```
828.46507522*2020-1632210.7578554575
```

Out[21]:
```
41288.694088942604
```

Predict income in 2024

In [22]:
```
prediction_2020 = regression.predict([[2024]])

print(f"Predicted income for 2024: ${prediction_2020[0]:,.2f}")
```

```
Predicted income for 2024: $44,602.55
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

In [23]:
```
hp = pd.DataFrame({'area': [2600,3000,3200,3600,4000],
                   'bedrooms': [3,4,3,3,5],
                   'age':[20,15,18,30,8],
       'price': [550000, 565000,610000,595000,760000]
       })
hp
```

Out[23]:

	area	bedrooms	age	price
0	2600	3	20	550000
1	3000	4	15	565000
2	3200	3	18	610000
3	3600	3	30	595000
4	4000	5	8	760000

In [24]:
```
import math
```

In [25]:
```
median_bedrooms = math.floor(hp.bedrooms.median())
median_bedrooms
```

Out[25]:
```
3
```

reg_1 = linear_model.LinearRegression(): This creates a new instance of the linear regression model.

reg_1.fit(hp[['area','bedrooms','age']], hp.price): This fits the model using the hp DataFrame.

hp[['area', 'bedrooms', 'age']]: These are the independent variables (features) used for prediction: house size (area), number of bedrooms (bedrooms), and house age (age).

hp.price: This is the dependent variable (target) that the model will try to predict: house price (price).

In [26]:
```
reg_1 = linear_model.LinearRegression()
reg_1.fit(hp[['area','bedrooms','age']],hp.price)
```

Out[26]:
```
▾ LinearRegression
LinearRegression()
```

coef_:m, mx+b

In [27]:
```
reg_1.coef_
```

Out[27]:
```
array([   137.25, -26025.  ,  -6825.  ])
```

intercept:b

In [28]:
```
reg_1.intercept_
```

Out[28]:
```
383724.9999999998
```

In [29]:
```
prediction_price = reg_1.predict([[3000,3,40]])

print(f"Predicted price: ${prediction_price[0]:,.2f}")
```

```
Predicted price: $444,400.00
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

In [30]:
```
137.25*3000+-26025*3+-6825*40+383724.9999999998
```

Out[30]:
```
444399.9999999998
```

In [31]:
```
prediction_price_1= reg_1.predict([[2500,4,5]])

print(f'Predicted Price: ${prediction_price_1[0]:,.2f}')
```

```
Predicted Price: $588,625.00
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

In [32]:
```
hi_sal = pd.DataFrame({
    'experience':[0,0,5,2,7,3,10,11],
    'test_score':[8,8,6,10,9,7,7,7],
    'interview_score':[9,6,7,10,6,10,7,8],
    'salary':[50000,45000,60000,65000,70000,62000,72000,80000]
})
hi_sal
```

Out[32]:

	experience	test_score	interview_score	salary
0	0	8	9	50000
1	0	8	6	45000
2	5	6	7	60000
3	2	10	10	65000
4	7	9	6	70000
5	3	7	10	62000
6	10	7	7	72000
7	11	7	8	80000

In [33]:
```
import math
median_test_score = math.floor(hi_sal['test_score'].mean())
median_test_score
```

Out[33]:
```
7
```

Fit the Model

In [34]:
```
reg_2 = linear_model.LinearRegression()
reg_2.fit(hi_sal[['experience','test_score','interview_score']], hi_sal.salary
```

Out[34]:
```
LinearRegression()
```

2 years of experience

Test score of 9

Interview score of 6

In [35]:
```
predicted_salary = reg_2.predict([[2,9,6]])

print(f'Predicted Salary: ${predicted_salary[0]:,.2f}')
```

```
Predicted Salary: $53,713.87
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

m, mx+b

In [36]:
```
reg_2.coef_
```

Out[36]:
```
array([2922.26901502, 2221.30909959, 2147.48256637])
```

b, mx+b

In [37]:
```
reg_2.intercept_
```

Out[37]:
```
14992.65144669314
```

(mx+mx+mx)+b

In [38]:
```
2922.26901502*2+2221.30909959*9+2147.48256637*6+14992.65144669314
```

Out[38]:
```
53713.86677126314
```

In [39]:
```
predicted_salary_1 = reg_2.predict([[12,10,10]])

print(f'Predicted Salary: ${predicted_salary_1[0]:,.2f}')
```

```
Predicted Salary: $93,747.80
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X do
es not have valid feature names, but LinearRegression was fitted with feature
names
  warnings.warn(
```

In [40]:
```
2922.26901502*12+2221.30909959*10+2147.48256637*10+14992.65144669314
```

Out[40]:
```
93747.79628653315
```

In [ ]:
```

```