📈 Linear Regression

In Machine Learning and this notebook we use Scikit-learn a lot.

What is scikit-learn used for?

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python.

What is linear regression used for?

Linear regression analysis is used to predict the value of a variable based on the value of another variable. The variable you want to predict is called the dependent variable. The variable you are using to predict the other variable's value is called the independent variable.

Making Predictions with Linear Regression

Given the representation is a linear equation, making predictions is as simple as solving the equation for a specific set of inputs.

Let’s make this concrete with an example. Imagine we are predicting weight (y) from height (x). Our linear regression model representation for this problem would be:

y = B0 + B1 * x1

or

weight =B0 +B1 * height

Where B0 is the bias coefficient and B1 is the coefficient for the height column. We use a learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight.

For example, lets use B0 = 0.1 and B1 = 0.5. Let's plug them in and calculate the weight (in kilograms) for a person with the height of 182 centimeters.

weight = 0.1 + 0.5 * 182

weight = 91.1

You can see that the above equation could be plotted as a line in two-dimensions. The B0 is our starting point regardless of what height we have. We can run through a bunch of heights from 100 to 250 centimeters and plug them to the equation and get weight values, creating our line.

Now that we know how to make predictions given a learned linear regression model, let's look at some rules of thumb for preparing our data to make the most of this type of model.

📤 Import & Install Libraries

In [1]:
```
!pip install hvplot
```

```
Collecting hvplot
  Downloading hvplot-0.7.3-py2.py3-none-any.whl (3.1 MB)
     |████████████████████████████████| 3.1 MB 884 kB/s
Requirement already satisfied: bokeh>=1.0.0 in /opt/conda/lib/python3.7/site-packages (from hvplot) (2.3.2)
Requirement already satisfied: holoviews>=1.11.0 in /opt/conda/lib/python3.7/site-packages (from hvplot) (1.14.4)
Requirement already satisfied: pandas in /opt/conda/lib/python3.7/site-packages (from hvplot) (1.2.4)
Requirement already satisfied: colorcet>=2 in /opt/conda/lib/python3.7/site-packages (from hvplot) (2.0.6)
Requirement already satisfied: numpy>=1.15 in /opt/conda/lib/python3.7/site-packages (from hvplot) (1.19.5)
Requirement already satisfied: pillow>=7.1.0 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvplot) (8.2.0)
Requirement already satisfied: python-dateutil>=2.1 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvplot)
(2.8.1)
Requirement already satisfied: typing-extensions>=3.7.4 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvpl
ot) (3.7.4.3)
Requirement already satisfied: PyYAML>=3.10 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvplot) (5.4.1)
Requirement already satisfied: packaging>=16.8 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvplot) (20.
9)
Requirement already satisfied: tornado>=5.1 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvplot) (6.1)
Requirement already satisfied: Jinja2>=2.9 in /opt/conda/lib/python3.7/site-packages (from bokeh>=1.0.0->hvplot) (3.0.1)
Requirement already satisfied: param>=1.7.0 in /opt/conda/lib/python3.7/site-packages (from colorcet>=2->hvplot) (1.11.1)
Requirement already satisfied: pyct>=0.4.4 in /opt/conda/lib/python3.7/site-packages (from colorcet>=2->hvplot) (0.4.8)
Requirement already satisfied: panel>=0.8.0 in /opt/conda/lib/python3.7/site-packages (from holoviews>=1.11.0->hvplot) (0.
11.3)
Requirement already satisfied: pyviz-comms>=0.7.4 in /opt/conda/lib/python3.7/site-packages (from holoviews>=1.11.0->hvplo
t) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /opt/conda/lib/python3.7/site-packages (from Jinja2>=2.9->bokeh>=1.0.0->
hvplot) (2.0.1)
Requirement already satisfied: pyparsing>=2.0.2 in /opt/conda/lib/python3.7/site-packages (from packaging>=16.8->bokeh>=1.
0.0->hvplot) (2.4.7)
Requirement already satisfied: pytz>=2017.3 in /opt/conda/lib/python3.7/site-packages (from pandas->hvplot) (2021.1)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.7/site-packages (from panel>=0.8.0->holoviews>=1.11.0->hvplo
t) (4.61.1)
Requirement already satisfied: markdown in /opt/conda/lib/python3.7/site-packages (from panel>=0.8.0->holoviews>=1.11.0->h
vplot) (3.3.4)
Requirement already satisfied: requests in /opt/conda/lib/python3.7/site-packages (from panel>=0.8.0->holoviews>=1.11.0->h
vplot) (2.25.1)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.1->bokeh>=1.0.0
->hvplot) (1.15.0)
Requirement already satisfied: importlib-metadata in /opt/conda/lib/python3.7/site-packages (from markdown->panel>=0.8.0->
holoviews>=1.11.0->hvplot) (3.4.0)
Requirement already satisfied: zipp>=0.5 in /opt/conda/lib/python3.7/site-packages (from importlib-metadata->markdown->pan
el>=0.8.0->holoviews>=1.11.0->hvplot) (3.4.1)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/conda/lib/python3.7/site-packages (from requests->panel>=0.8.
0->holoviews>=1.11.0->hvplot) (1.26.5)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.7/site-packages (from requests->panel>=0.8.0->
holoviews>=1.11.0->hvplot) (2021.5.30)
Requirement already satisfied: idna<3,>=2.5 in /opt/conda/lib/python3.7/site-packages (from requests->panel>=0.8.0->holovi
```

```
ews>=1.11.0->hvplot) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /opt/conda/lib/python3.7/site-packages (from requests->panel>=0.8.0->h
oloviews>=1.11.0->hvplot) (4.0.0)
Installing collected packages: hvplot
Successfully installed hvplot-0.7.3
WARNING: Running pip as root will break packages and permissions. You should install packages reliably by using venv: http
s://pip.pypa.io/warnings/venv
```

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import hvplot.pandas

from sklearn.model_selection import train_test_split

from sklearn import metrics

from sklearn.linear_model import LinearRegression

%matplotlib inline
```

💾 Check out the Data

In [3]:
```
df=pd.read_csv('../input/real-estate-price-prediction/Real estate.csv')
```

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

Out[4]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
0	1	2012.917	32.0	84.87882	10	24.98298	121.54024	37.9
1	2	2012.917	19.5	306.59470	9	24.98034	121.53951	42.2
2	3	2013.583	13.3	561.98450	5	24.98746	121.54391	47.3
3	4	2013.500	13.3	561.98450	5	24.98746	121.54391	54.8
4	5	2012.833	5.0	390.56840	5	24.97937	121.54245	43.1

In [5]:
```
df.shape
```

Out[5]:
```
(414, 8)
```

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   No                                      414 non-null    int64  
 1   X1 transaction date                     414 non-null    float64
 2   X2 house age                            414 non-null    float64
 3   X3 distance to the nearest MRT station  414 non-null    float64
 4   X4 number of convenience stores         414 non-null    int64  
 5   X5 latitude                             414 non-null    float64
 6   X6 longitude                            414 non-null    float64
 7   Y house price of unit area              414 non-null    float64
dtypes: float64(6), int64(2)
memory usage: 26.0 KB
```

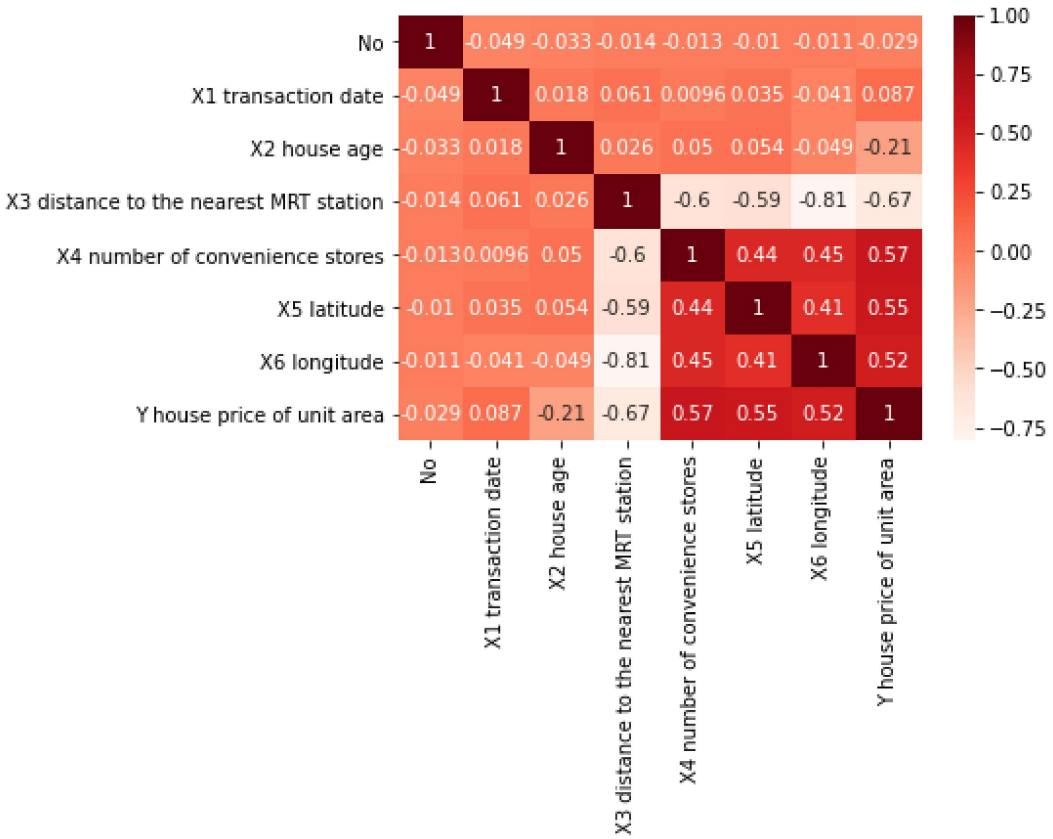
In [7]:
```
df.corr()
```

Out[7]:

	No	X1 transaction date	X2 house age	X3 distance to the nearest MRT station	X4 number of convenience stores	X5 latitude	X6 longitude	Y house price of unit area
No	1.000000	-0.048658	-0.032808	-0.013573	-0.012699	-0.010110	-0.011059	-0.028587
X1 transaction date	-0.048658	1.000000	0.017549	0.060880	0.009635	0.035058	-0.041082	0.087491
X2 house age	-0.032808	0.017549	1.000000	0.025622	0.049593	0.054420	-0.048520	-0.210567
X3 distance to the nearest MRT station	-0.013573	0.060880	0.025622	1.000000	-0.602519	-0.591067	-0.806317	-0.673613
X4 number of convenience stores	-0.012699	0.009635	0.049593	-0.602519	1.000000	0.444143	0.449099	0.571005
X5 latitude	-0.010110	0.035058	0.054420	-0.591067	0.444143	1.000000	0.412924	0.546307
X6 longitude	-0.011059	-0.041082	-0.048520	-0.806317	0.449099	0.412924	1.000000	0.523287
Y house price of unit area	-0.028587	0.087491	-0.210567	-0.673613	0.571005	0.546307	0.523287	1.000000

In [8]:
```
sns.heatmap(df.corr(), annot=True,cmap='Reds')
```

Out[8]:
```
<AxesSubplot:>
```



📊 Exploratory Data Analysis (EDA)

In [9]:
```
sns.pairplot(df)
```

Out[9]:
```
<seaborn.axisgrid.PairGrid at 0x7fcf18e2e910>
```


No
X1 transaction date
X2 house age
X3 distance to the nearest MRT station
X4 number of convenience stores
X5 latitude
X6 longitude

📈 Training a Linear Regression Model

X and y arrays

In [10]:
```
X=df.drop('Y house price of unit area', axis=1)

y=df['X4 number of convenience stores']
```

In [11]:
```
print("X=",X.shape,"\ny=", y.shape)
```

```
X= (414, 7)
y= (414,)
```

🧱 Train Test Split

Now let's split the data into a training set and a testing set. We will train out model on the training set and then use the test set to evaluate the model.

In [12]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [13]:
```
X_train.shape
```

Out[13]:
```
(289, 7)
```

In [14]:
```
X_test.shape
```

Out[14]:
```
(125, 7)
```

✔️ Linear Regression

In [15]:
```
model = LinearRegression()
```

In [16]:
```
model.fit(X_train, y_train)
```

Out[16]:
```
LinearRegression()
```

✔ Model Evaluation

In [17]:
```
model.coef_
```

Out[17]:
```
array([-1.26803420e-17, -5.07334605e-15, -5.99955923e-17,  7.07810166e-18,
        1.00000000e+00,  1.09793322e-15, -1.38744293e-15])
```

In [18]:
```
pd.DataFrame(model.coef_, X.columns, columns=['Coedicients'])
```

Out[18]:

	Coedicients
No	-1.268034e-17
X1 transaction date	-5.073346e-15
X2 house age	-5.999559e-17
X3 distance to the nearest MRT station	7.078102e-18
X4 number of convenience stores	1.000000e+00
X5 latitude	1.097933e-15
X6 longitude	-1.387443e-15

✔ Predictions from our Model

In [19]:
```
y_pred = model.predict(X_test)
```

✔ Regression Evaluation Metrics

Here are three common evaluation metrics for regression problems:

- **Mean Absolute Error** (MAE) is the mean of the absolute value of the errors:

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

- **Mean Squared Error** (MSE) is the mean of the squared errors:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- **Root Mean Squared Error** (RMSE) is the square root of the mean of the squared errors:

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

📌 Comparing these metrics:

- **MAE** is the easiest to understand, because it's the average error.
- **MSE** is more popular than MAE, because MSE "punishes" larger errors, which tends to be useful in the real world.
- **RMSE** is even more popular than MSE, because RMSE is interpretable in the "y" units.

All of these are **loss functions**, because we want to minimize them.

In [20]:
```
MAE= metrics.mean_absolute_error(y_test, y_pred)
MSE=metrics.mean_squared_error(y_test, y_pred)
RMSE= np.sqrt(MSE)
```

In [21]:
```
MAE
```

Out[21]:
```
8.536346855515258e-15
```

In [22]:
```
MSE
```

Out[22]:
```
1.2905495539452208e-28
```

In [23]:
```
RMSE
```

Out[23]:
```
1.1360235710341668e-14
```

In [24]:
```
df['X4 number of convenience stores'].mean()
```

Out[24]:
```
4.094202898550725
```

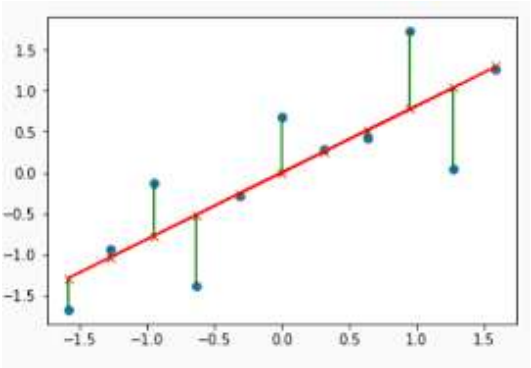
Residual Histogram

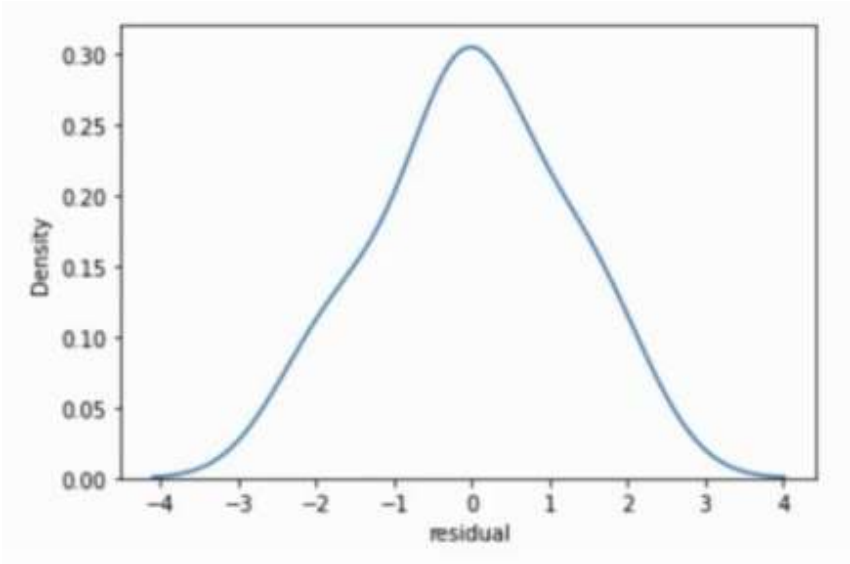
- **Often for Linear Regression it is a good idea to separately evaluate residuals**

$$(y - \hat{y})$$

  and not just calculate performance metrics (e.g. RMSE).

- **Let's explore why this is important...**
- **The residual eerors should be random and close to a normal distribution.**

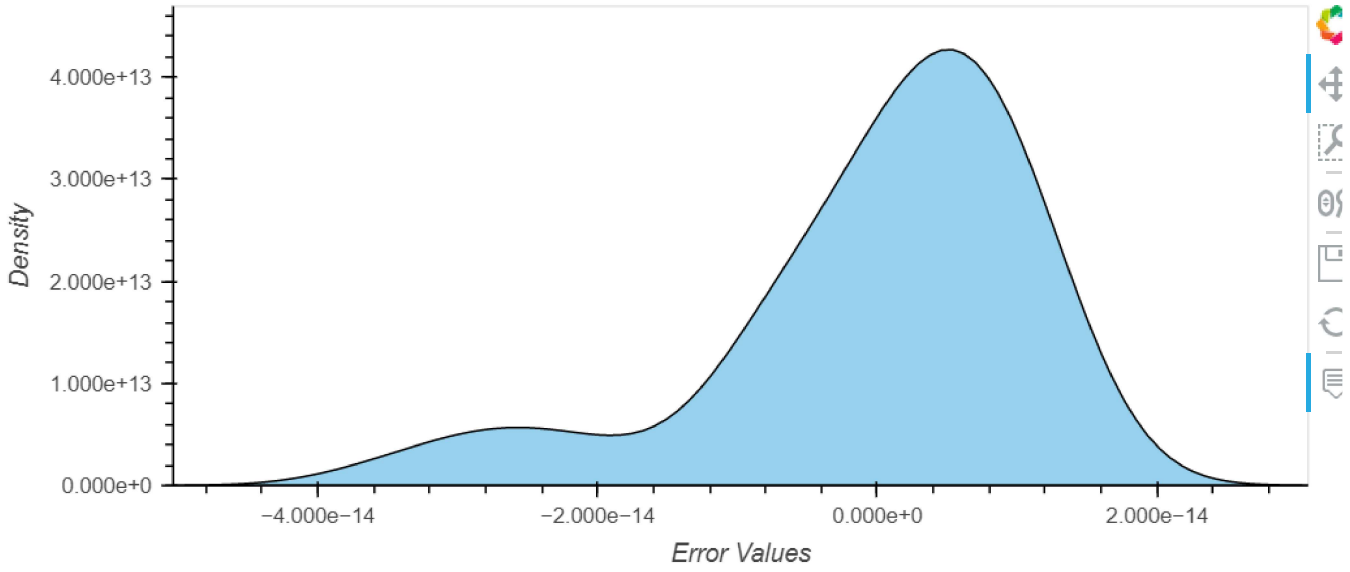




In [25]:
```
test_residual= y_test - y_pred
```

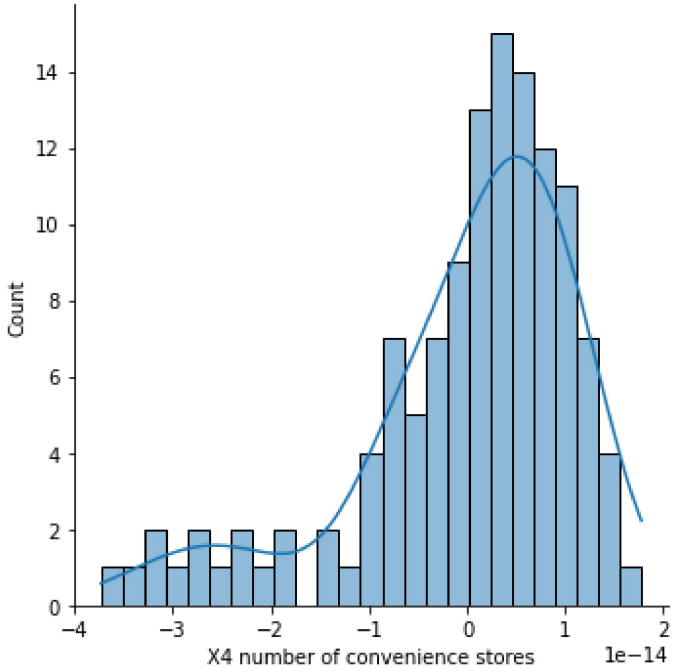
In [26]:
```
pd.DataFrame({'Error Values': (test_residual)}).hvplot.kde()
```

Out[26]:



In [27]:
```
sns.displot(test_residual, bins=25, kde=True)
```

Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x7fcef584d6d0>
```

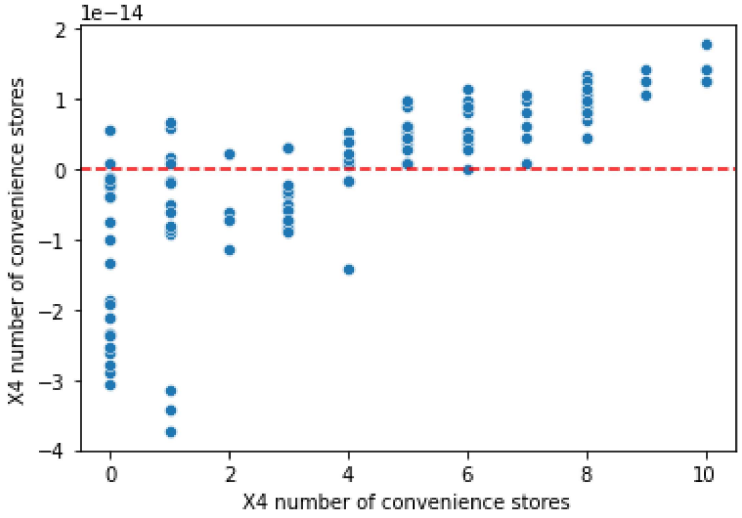


- **Residual plot shows residual error VS. true y value.**

In [28]:
```
sns.scatterplot(x=y_test, y=test_residual)

plt.axhline(y=0, color='r', ls='--')
```

Out[28]:
```
<matplotlib.lines.Line2D at 0x7fcf0697bc50>
```



- **Residualplot showing a clear pattern, indicating Linear Regression no valid!**

Finished, but you can copy this notebook and start practicing.