Name: Jann Moises Nyll B. De los Reyes

Section: CPE22S3

Date: March 11, 2024

Submitted to: Engr. Roman M. Richard

📈 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 modelling 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 value of a variable based on the value of another variable. The variable you want to predict is called `dependent variable`. The variable you are using to predict the other variable's value is called the `independent variable`.

Making Prediction with Linear Regression

Given the representation is a linear equation, making prediction 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`

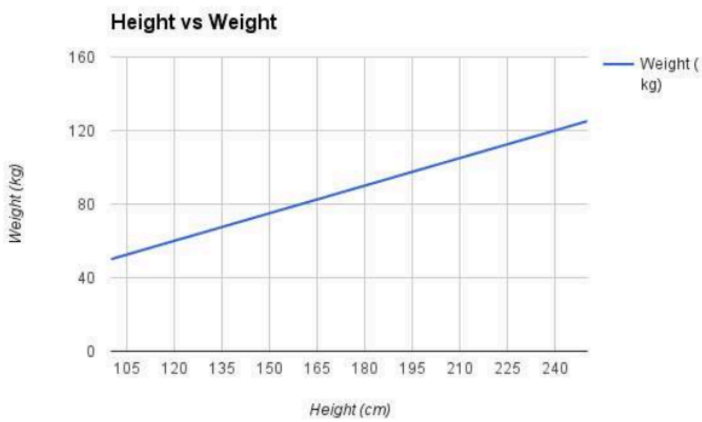
Where B0 is the *bias coefficient* and B1 is the coefficient for the height column. We use learning technique to find a good set of coefficient values. Once found, we can plug in different height values to predict the weight

For example, let's 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 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

```
!pip install hvplot
```

```
Requirement already satisfied: hvplot in /usr/local/lib/python3.10/dist-packages (0.9.2)
Requirement already satisfied: bokeh>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (3.4.1)
Requirement already satisfied: colorcet>=2 in /usr/local/lib/python3.10/dist-packages (from hvplot) (3.1.0)
Requirement already satisfied: holoviews>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (1.17.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from hvplot) (2.0.3)
Requirement already satisfied: numpy>=1.15 in /usr/local/lib/python3.10/dist-packages (from hvplot) (1.25.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from hvplot) (24.0)
Requirement already satisfied: panel>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (1.4.2)
Requirement already satisfied: param<3.0,>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from hvplot) (2.1.0)
Requirement already satisfied: Jinja2>=2.9 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (3.1.3)
Requirement already satisfied: contourpy>=1.2 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (1.2.1)
Requirement already satisfied: pillow>=7.1.0 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (9.4.0)
Requirement already satisfied: PyYAML>=3.10 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (6.0.1)
Requirement already satisfied: tornado>=6.2 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (6.3.3)
Requirement already satisfied: xyzservices>=2021.09.1 in /usr/local/lib/python3.10/dist-packages (from bokeh>=1.0.0->hvplot) (2024.4.0)
Requirement already satisfied: pyviz-comms>=0.7.4 in /usr/local/lib/python3.10/dist-packages (from holoviews>=1.11.0->hvplot) (3.0.2)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->hvplot) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->hvplot) (2023.4)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas->hvplot) (2024.1)
Requirement already satisfied: markdown in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (3.6)
Requirement already satisfied: markdown-it-py in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (3.0.0)
Requirement already satisfied: linkify-it-py in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (2.0.3)
Requirement already satisfied: mdit-py-plugins in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (0.4.0)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (2.31.0)
Requirement already satisfied: tqdm>=4.48.0 in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (4.66.2)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (6.1.0)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from panel>=0.11.0->hvplot) (4.11.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9->bokeh>=1.0.0->hvplot) (2.1.5)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->hvplot) (1.16.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->panel>=0.11.0->hvplot) (0.5.1)
Requirement already satisfied: uc-micro-py in /usr/local/lib/python3.10/dist-packages (from linkify-it-py->panel>=0.11.0->hvplot) (1.0.3)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py->panel>=0.11.0->hvplot) (0.1.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->panel>=0.11.0->hvplot) (2024.2.2)
```

```
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

```
df = pd.read_csv('/content/drive/MyDrive/Module 11/Real estate.csv')
```

```
df.head()
```

	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

Next steps: View recommended plots

```
1 df.shape
```

```
(414, 8)
```

```
1 df.info()
```

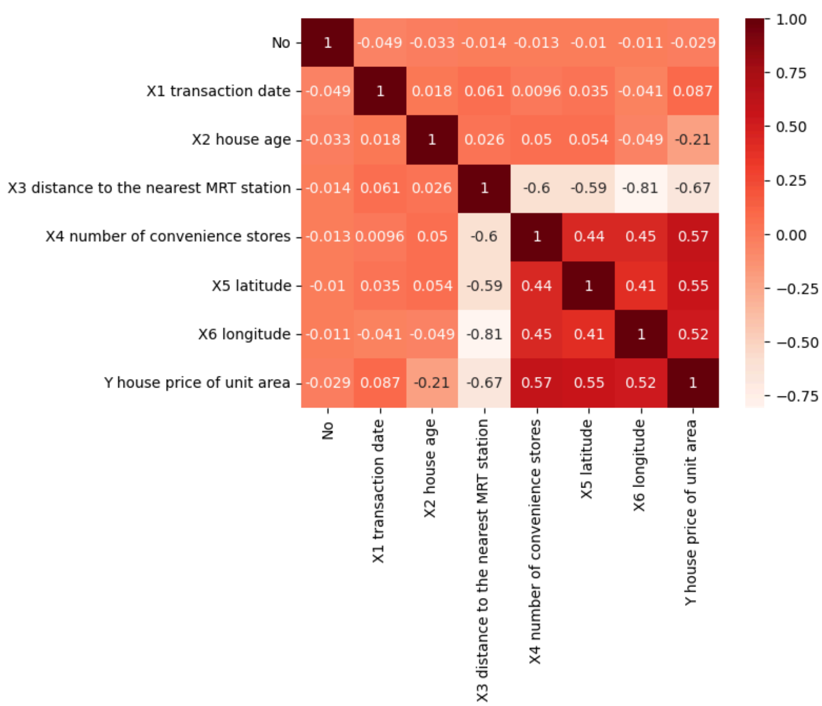
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 414 entries, 0 to 413
Data columns (total 8 columns):
 #   Column                                  Non-Null Count  Dtype  
---  ------                                  --------------  -----  
 0   No                                      414 non-null    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
```

```
1 df.corr()
```

	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

```
1 sns.heatmap(df.corr(),annot = True, cmap ='Reds')
```

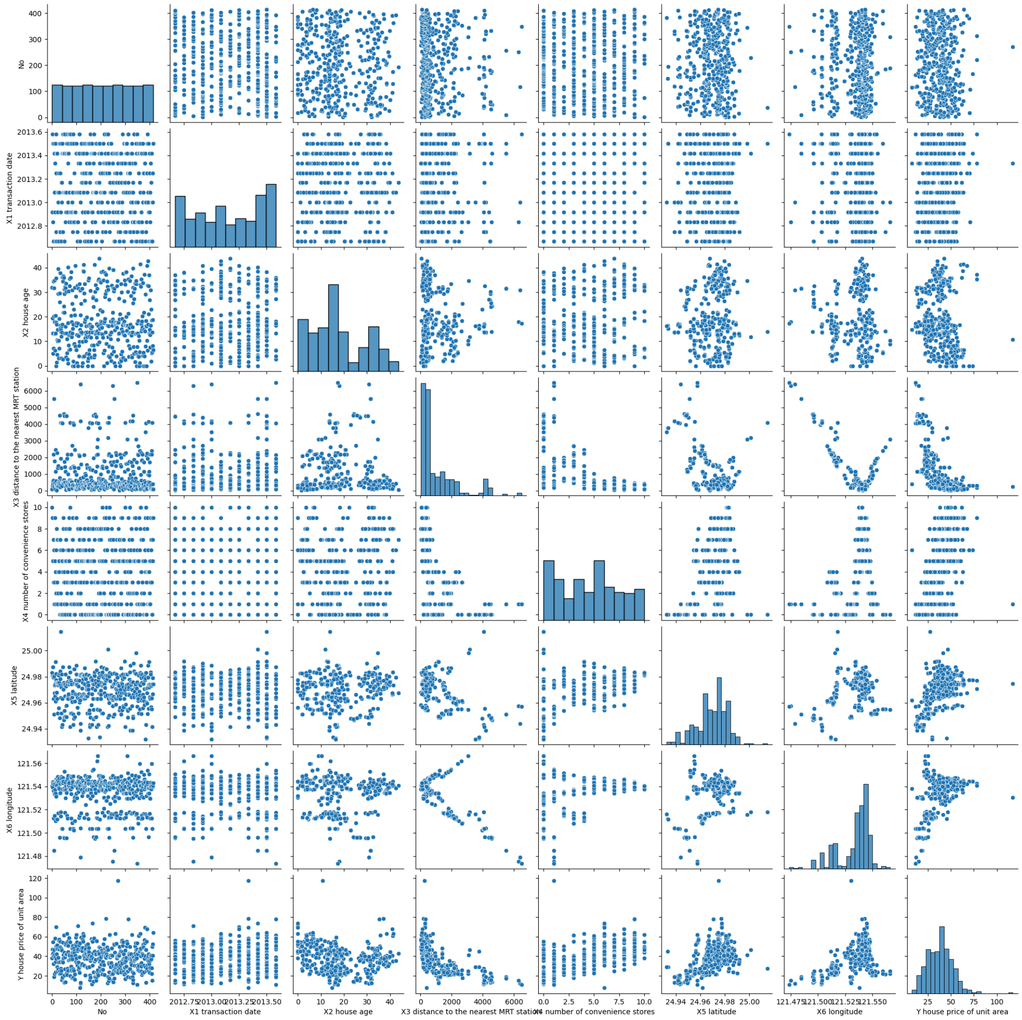
```
<Axes: >
```



📊 Explanatory Data Analysis (EDA)

```
1 sns.pairplot(df)
```

```
<seaborn.axisgrid.PairGrid at 0x7a9c0fda8550>
```



📈 Training a Linear Regression Model

X and Y arrays

```
1 X = df.drop('Y house price of unit area', axis = 1)
2 y = df['X4 number of convenience stores']
```

```
1 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.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 101)
```

```
X_train.shape
```

```
(289, 7)
```

```
X_test.shape
```

```
(125, 7)
```

Linear Regression

```
model = LinearRegression()
```

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

```
▾ LinearRegression
LinearRegression()
```

Model Evaluation

```
model.coef_
```

```
array([-1.49344835e-17, -9.09342046e-15, -1.36338423e-16,  1.73472348e-18,
        1.00000000e+00,  1.28927721e-14,  1.08238203e-14])
```

```
pd.DataFrame(model.coef_, X.columns, columns = ['Coefficients'])
```

	Coefficients
No	-1.493448e-17
X1 transaction date	-9.093420e-15
X2 house age	-1.363384e-16
X3 distance to the nearest MRT station	1.734723e-18
X4 number of convenience stores	1.000000e+00
X5 latitude	1.289277e-14
X6 longitude	1.082382e-14

Predictions from our Model

```
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 mean of the squared errors:
> $$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

📌 Comparing these metrics:

- **MAE** is ther easiest to understand, because it's the average error.
- **MSE** is more popular than MAE,because MSE "punishes" larger error, which tend 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.

```
MAE = metrics.mean_absolute_error(y_test, y_pred)
MSE = metrics.mean_squared_error(y_test, y_pred)
RMSE = np.sqrt(MSE)
```

```
MAE
```

```
4.231748536250847e-15
```

```
MSE
```

```
2.718688400256278e-29
```

```
RMSE
```

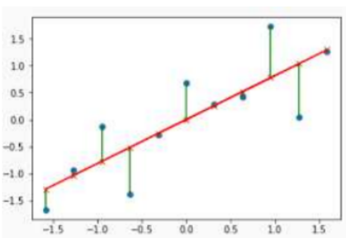
```
5.214104333685967e-15
```

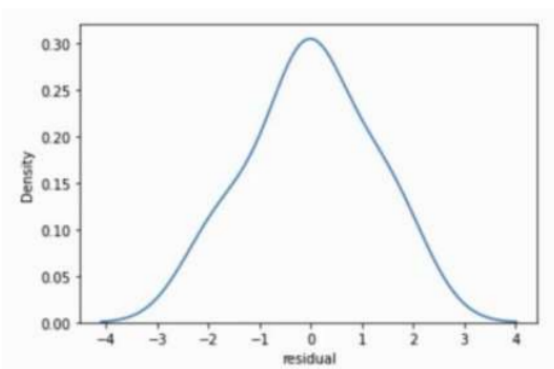
```
df['X4 number of convenience stores'].mean()
```

```
4.094202898550725
```

Residual Historgram

- Often for Linear Regression it is a good idea to separately evaluate residuals
$$(y - \hat{y})$$
and not just calculate the performance metrics(e.g. RMSE)
- Let's explore why this is important ...
- The residual errors should be random and close to a normal distribution

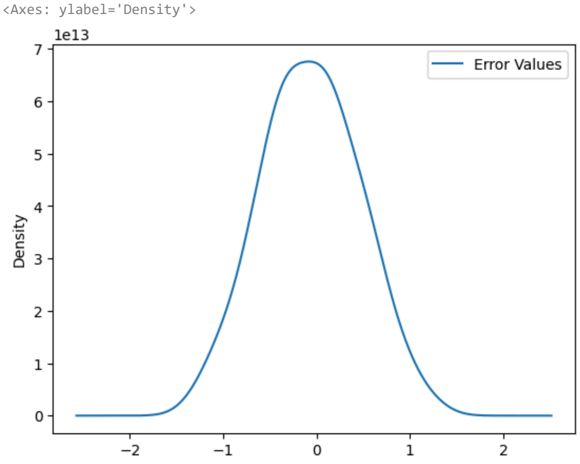





```
test_residual = y_test - y_pred
```

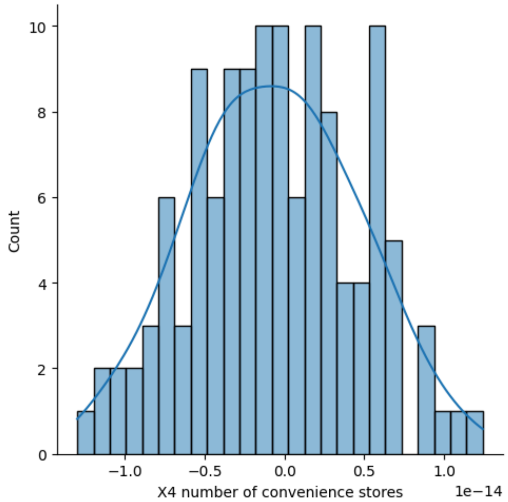
```
pd.DataFrame({'Error Values':(test_residual)}).plot.kde()
```

```
<Axes: ylabel='Density'>
```



```
sns.displot(test_residual, bins = 25 , kde =True)
```

```
<seaborn.axisgrid.FacetGrid at 0x7a9c0ba5a8c0>
```

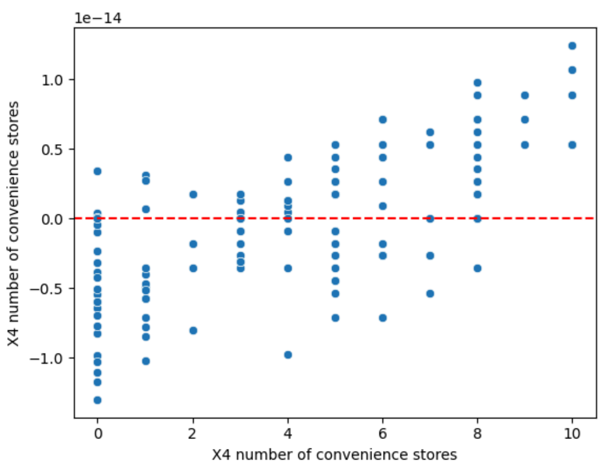


- Residual plot shows residual error VS. true y value

```
sns.scatterplot(x=y_test,y =test_residual)

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

```
<matplotlib.lines.Line2D at 0x7a9c08a67fa0>
```



- Residual plot showing clear pattern, indicating Linear Regression is no valid!