Portofolio Data Science

Medical Cost Personal Prediction

My portfolio as a data scientist. This project discusses the analysis of "Medical Cost Personal" data which I will explore more deeply which will later predict a person's personal health costs. So that through the data we can see how the data visualization, the relationship between the data, and the influence of the parameters related to a person's personal health costs.

The Purpose of This Project

I want to know what parameters or variables affect a person's personal health costs, try to visualize those parameters, and predict the estimated personal health costs of a person based on these parameters.

Library Used

In this project I use several libraries including,

1. Pandas
2. Numpy
3. Matplotlib
4. Seaborn
5. Tensorflow
6. Sklern

Dataset Used

The dataset I used can be downloaded for free on the kaggle website.

Medical Cost Personal Datasets | Kaggle

Tools Used

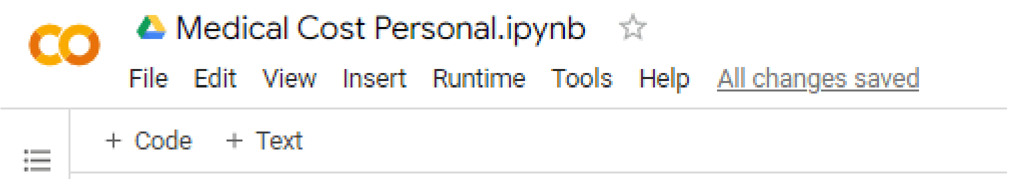
The tools I use are,

1. Google colab.
2. Google Drive.
3. Python.

Connect Google Colab with Google Drive

- **Create a new project**

I created a new project at Google Colab called "Medical Cost Personal Prediction". This project will be automatically saved in the cloud storage provided by Google Drive. The google account that we use when logging in to Google Colab will be the same as the Google Drive account storage.



- **Then we will connect with Google Drive,**

```
[1] from google.colab import drive

[2] drive.mount('/content/drive/')

    Mounted at /content/drive/
```

Load Dataset

- When we load the dataset we will get something like this,

```
data_insurance.head()
```

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

- Content in Dataset Medical Cost Personal

  Columns

- **age**: age of primary beneficiary
- **sex**: insurance contractor gender, female, male
- **bmi**: Body mass index, providing an understanding of body, weights that are relatively high or low relative to height, objective index of body weight (kg / m ^ 2) using the ratio of height to weight, ideally 18.5 to 24.9
- **children**: Number of children covered by health insurance / Number of dependents
- **smoker**: Smoking
- **region**: the beneficiary's residential area in the US, northeast, southeast, southwest, northwest.
- **charges**: Individual medical costs billed by health insurance

Exploratory Data Analysis

1. **What is the length of the data and in what row is the last data?**

	age	sex	bmi	children	smoker	region	charges
1333	50	male	30.97	3	no	northwest	10600.5483
1334	18	female	31.92	0	no	northeast	2205.9808
1335	18	female	36.85	0	no	southeast	1629.8335
1336	21	female	25.80	0	no	southwest	2007.9450
1337	61	female	29.07	0	yes	northwest	29141.3603

In the dataset, it turns out that the last data is in the 1337th array because it starts from 0 then the last line is 1338, where the age of the insurance user is 61, gender is female, BMI 29.07, number of children is 0, northwest area, and is charged 29141.3603.

2. How many rows and columns does the dataset have?

```
(1338, 7)
```

The dataset contains 1338 rows and 7 columns.

3. What is the size of the dataset?

```
9366
```

the size of the dataset is 1338 x 7 = 9366.

4. What is the title of each table?

```
data_insurance.columns
```

```
Index(['age', 'sex', 'bmi', 'children', 'smoker', 'region', 'charges'], dtype='object')
```

In the dataset there are 7 columns, where each column is entitled "age", "sex", "bmi", "children", "smoker", "region", and "charges".

5. Is there any blank or null data in the dataset?

```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

wow... wonderfull!,

No blank or null data.

6. Let's see the data type!

```
[11] data_insurance.dtypes

age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

from the dataset can be seen the data type of the dataset. There are integer, object, and float types.

7. What if we see the info from the data?

you can see it!

```
[53] data_insurance.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       1338 non-null   int64  
 1   sex       1338 non-null   object 
 2   bmi       1338 non-null   float64
 3   children  1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

8. What if we look at the results of statistical calculations?

```
[13] data_insurance.describe()
```

	age	bmi	children	charges
count	1338.000000	1338.000000	1338.000000	1338.000000
mean	39.207025	30.663397	1.094918	13270.422265
std	14.049960	6.098187	1.205493	12110.011237
min	18.000000	15.960000	0.000000	1121.873900
25%	27.000000	26.296250	0.000000	4740.287150
50%	39.000000	30.400000	1.000000	9382.033000
75%	51.000000	34.693750	2.000000	16639.912515
max	64.000000	53.130000	5.000000	63770.428010

9. What is the unique data in the "age" column?

```
[21] data_insurance['age'].unique()

array([19, 18, 28, 33, 32, 31, 46, 37, 60, 25, 62, 23, 56, 27, 52, 30, 34,
       59, 63, 55, 22, 26, 35, 24, 41, 38, 36, 21, 48, 40, 58, 53, 43, 64,
       20, 61, 44, 57, 29, 45, 54, 49, 47, 51, 42, 50, 39])
```

10. What is the unique data in the "charges" column?

```
data_insurance['charges'].unique()

array([16884.924 ,  1725.5523,  4449.462 , ...,  1629.8335,  2007.945 ,
       29141.3603])
```

11.What is the unique data in the "children" column?

```
[23] data_insurance['children'].unique()

array([0, 1, 3, 2, 5, 4])
```

12. What is the unique data in the "region" column?

```
[24] data_insurance['region'].unique()
```

```
array(['southwest', 'southeast', 'northwest', 'northeast'], dtype=object)
```

13. What is the unique data in the "sex" column?

```
[25] data_insurance['sex'].unique()
```

```
array(['female', 'male'], dtype=object)
```

14. What is the unique data in the "smoker" column?

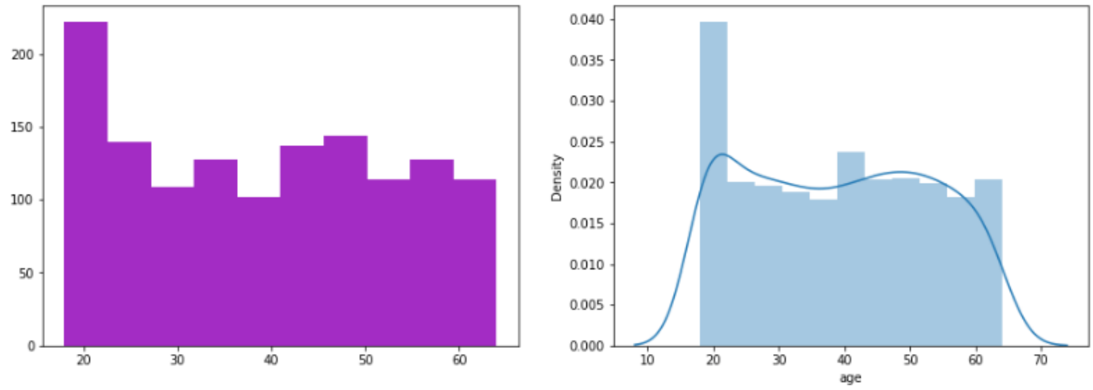
```
[26] data_insurance['smoker'].unique( )
```

```
array(['yes', 'no'], dtype=object)
```

Data Visualization

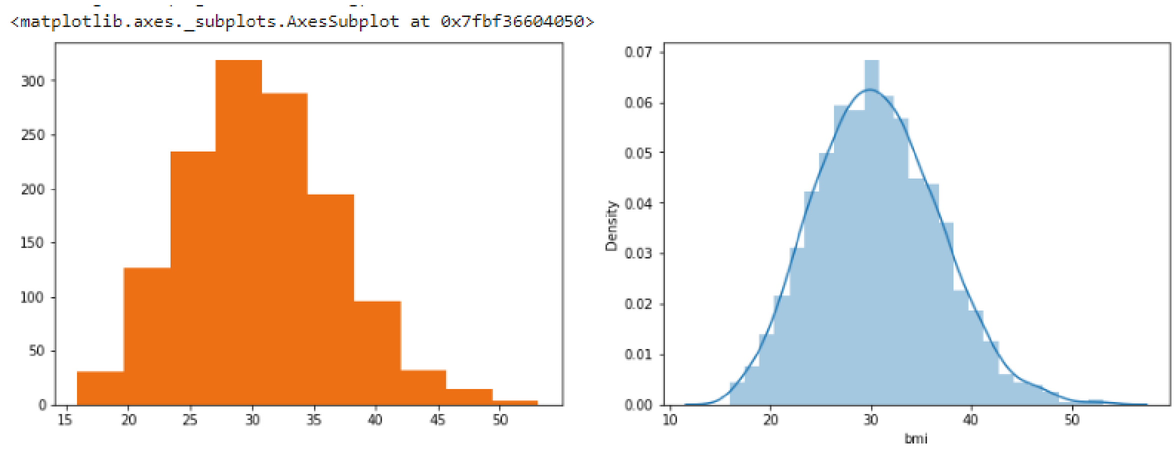
1. **At what age do users use health insurance a lot?**

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf3755ead0>
```



Health insurance users are dominated by users with a relatively young age of 20 years. The data frequency is more than 200 from 1338 users.

2. Do health insurance users have a normal body mass index (BMI)?

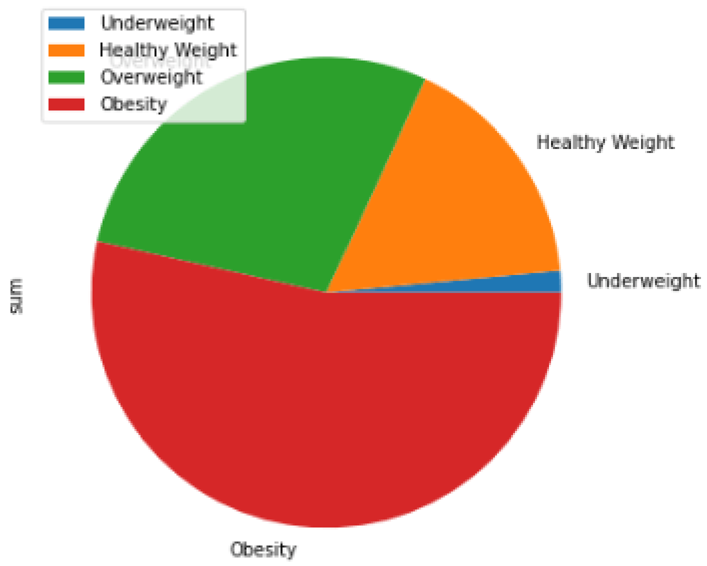


Visualization of the distribution of BMI data.

If viewed from the BMI guidelines then:

BMI	Weight status
Below 18.5	Underweight
18.5 – 24.9	Healthy Weight
25.0 – 29.9	Overweight
30.0 and Above	Obesity

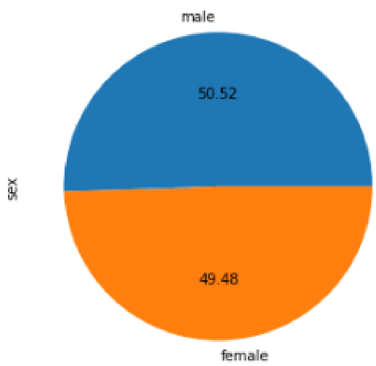
Based on the dataset we have and the person's BMI guidelines, we will visualize it like this:

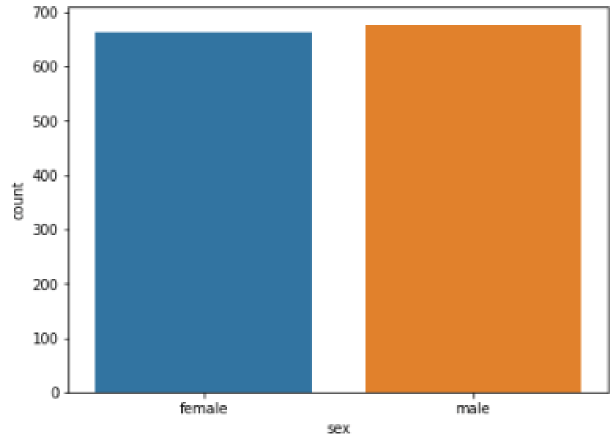


It turns out that when we look through the data visualization, the bottom line is that health insurance users are dominated Obesity or overweight status.

3. What about gender? approximately how many insurance users are female and male?

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf364cbf50>
```



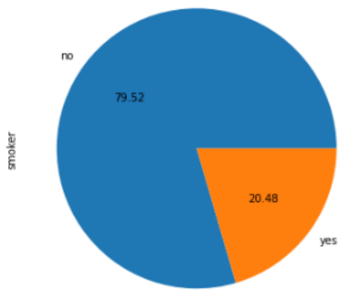


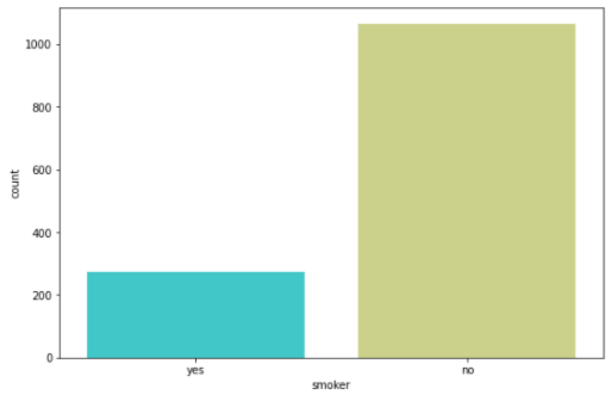
From these data, it can be seen that the number of insurance users by gender is almost the same and slightly dominated by male sex or about 50.52%.

- Male: 50.52 %
- Female: 49.48%

4. Smoking is one indicator of the cause of several serious diseases. What about health insurance users, are they smokers?

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf363de890>
```

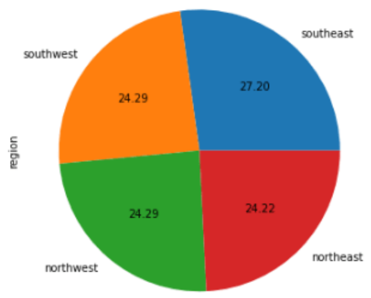


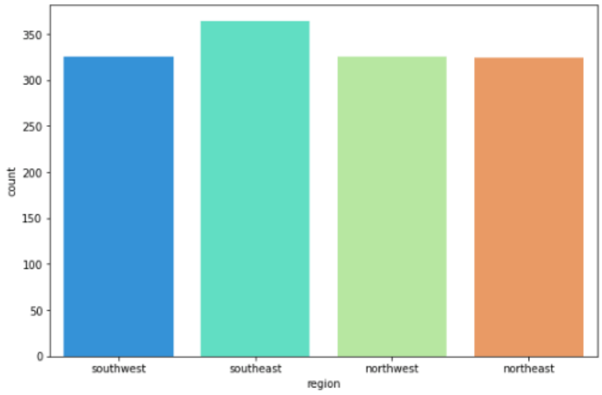


It turns out that the insurance user is more dominant than someone who is not a smoker.

5. How is the distribution area of each user?

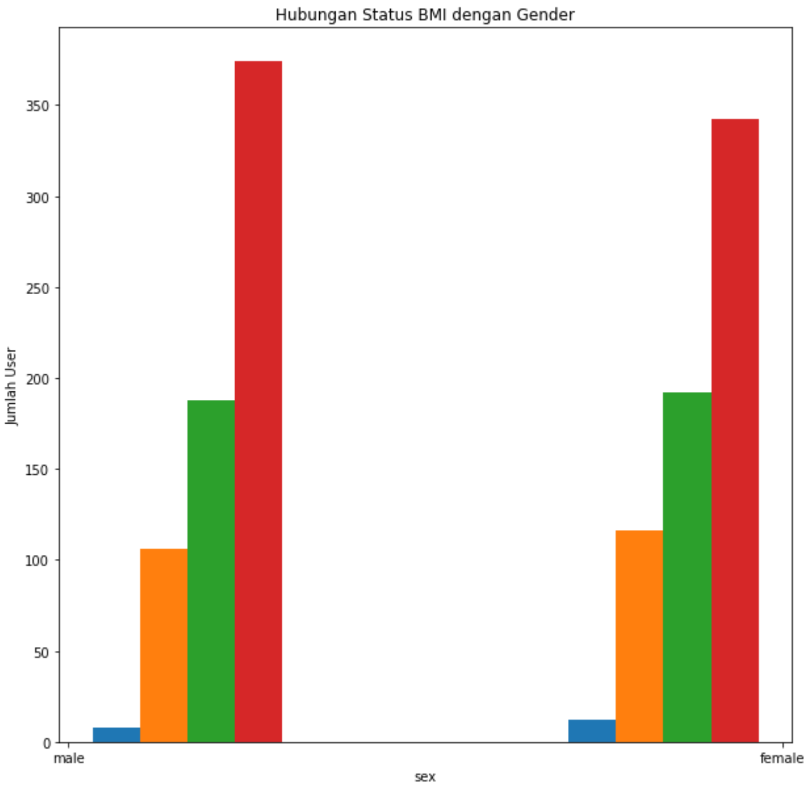
<matplotlib.axes._subplots.AxesSubplot at 0x7fbf36374110>





It turns out that the area distribution of each user is evenly distributed in 4 parts of the region, ranging from 24.22-27.20%.

6. when viewed from the BMI status guide. How many people have normal BMI status based on gender?



One-Hot Encoding

Because in the "Medical Cost Personal" dataset there is valuable data or object type, then we need to change the object data into number data so that when it is processed by the computer properly.

Why is it necessary to change object data to number? because the computer can only process data numbers.

To convert object data into number data, we will use the One-hot Encoding technique where the data value is 0 or 1.

How is the result?

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	region_southwest
0	19	27.900	0	16884.92400	1	0	0	1	0	0	0	1
1	18	33.770	1	1725.55230	0	1	1	0	0	0	1	0
2	28	33.000	3	4449.46200	0	1	1	0	0	0	1	0
3	33	22.705	0	21984.47061	0	1	1	0	0	1	0	0
4	32	28.880	0	3866.85520	0	1	1	0	0	1	0	0

Now the object data has changed to number data.

Preprocessing Data(Normalization and Standardization)

This data normalization stage is to change each data number into a data number whose values range from 0 to 1.

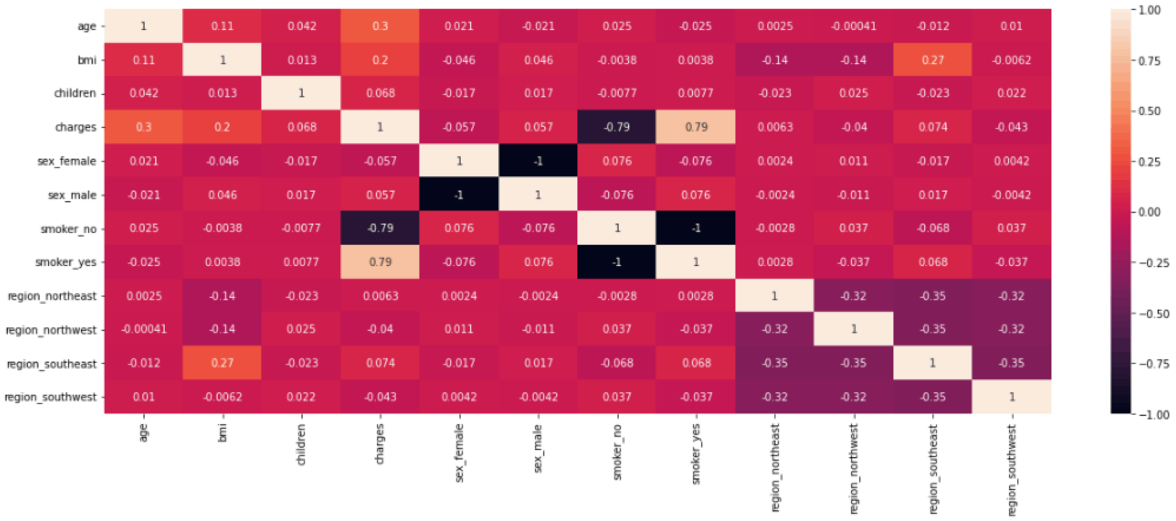
To compose it, we need a data normalization technique, by dividing the old data by the largest data in that column. then the results of the division into new data ranging from 0 to 1.

	age	bmi	children	charges	sex_female	sex_male	smoker_no	smoker_yes	region_northeast	region_northwest	region_southeast	region_southwest
0	0.296875	0.525127	0.0	0.264777	1.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
1	0.281250	0.635611	0.2	0.027059	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0
2	0.437500	0.621118	0.6	0.069773	0.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0
3	0.515625	0.427348	0.0	0.344744	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
4	0.500000	0.543572	0.0	0.060637	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1333	0.781250	0.582910	0.6	0.166230	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0
1334	0.281250	0.600791	0.0	0.034593	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0
1335	0.281250	0.693582	0.0	0.025558	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0
1336	0.328125	0.485601	0.0	0.031487	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0
1337	0.953125	0.547149	0.0	0.456973	1.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0

1338 rows × 12 columns

Data Correlation Visualization

We will look at the correlation between the data.



From the data visualization, it can be seen the correlation between the values of the columns and rows that represent information. We can see where:

1. correlation/relationship between insurance costs and age: 30%.
2. The correlation between insurance costs and BMI is: 20%.
3. the correlation between insurance costs and the number of children is: 6.8%.
4. The correlation between insurance costs and gender is approximately: 5.7 %.
5. correlation between insurance costs and whether he smokes or not is: 79%.
6. The correlation between insurance costs and the area is quite low around: 0 - 4%.

Split Data

```
from sklearn.model_selection import train_test_split

X = df_norm.drop(labels='charges', axis=1)
y = df_norm['charges']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((1070, 11), (268, 11), (1070,), (268,))
```

Because in this case we will predict a person's health costs based on related variables or parameters, we will use a linear regression algorithm.

Modeling

- **Linear Regresion from sklearn**

  Import Libraries for modeling.

```
[40] from sklearn.linear_model import LinearRegression

     lin_reg = LinearRegression()
```

Predict the model that has been made previously using test data.

```
y_pred = lin_reg.predict(X_test)
y_pred
```

```
array([ 0.14065376,  0.11084679,  0.57798594,  0.14826117,  0.42297307,
        0.17036287,  0.00267022,  0.26506722,  0.01713068,  0.17591764,
        0.44066953,  0.1470546 ,  0.08253135,  0.60241155,  0.63126162,
        0.58174697,  0.23898842,  0.56315888,  0.14289576,  0.49336224,
        0.06033656,  0.15885294,  0.03717306,  0.1119675 ,  0.17722584,
        0.20325493,  0.22752666,  0.09659489,  0.15624575,  0.03415152,
        0.14294928,  0.20501179,  0.0715351 ,  0.05344495,  0.06993545,
        0.20435907,  0.03104877,  0.13820329,  0.52173542,  0.51098161,
        0.06129426,  0.06783878,  0.22177699,  0.17913405,  0.13758947,
        0.18970048,  0.08282167,  0.04940471,  0.55659756,  0.1434852 ,
        0.24834153,  0.03675018,  0.19389527,  0.02324424,  0.20995721,
        0.19716935,  0.06808543,  0.50439268,  0.2088952 ,  0.20223826,
        0.22217185,  0.16474997,  0.25655756,  0.12174763,  0.18565424,
        0.06368465,  0.41794297,  0.17139828,  0.03351732,  0.09736506,
        0.16825705,  0.18234644,  0.17219651,  0.1437423 ,  0.18745801,
        0.10581474,  0.11366602,  0.16834084,  0.10319592,  0.13739916,
        0.05907337,  0.57444334,  0.10001689,  0.48364255,  0.54643705,
        0.55320428,  0.11007366,  0.20168258,  0.15590771,  0.22696298,
        0.2774542 ,  0.5528934 ,  0.51794524,  0.09700291,  0.50179983,
        0.14867913,  0.46171938,  0.05762052,  0.44390896,  0.09131764,
        0.08480055,  0.02953555,  0.18032927,  0.23640906,  0.18346484,
        0.06756775,  0.15516885,  0.49722734, -0.00136224,  0.51465413,
        0.05144535,  0.1596961 ,  0.22453611,  0.4961917 ,  0.17973187,
        0.06161535,  0.20554814,  0.49883615,  0.12783395,  0.05077721,
```

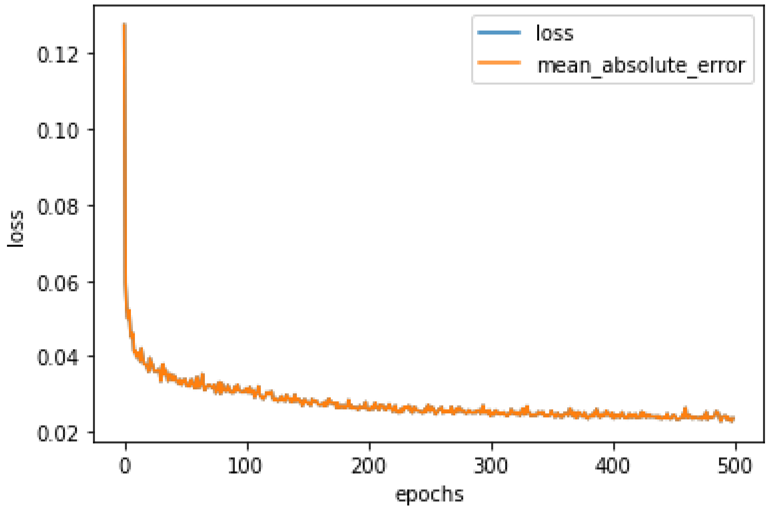
See the accuracy value of the model.

```
lin_reg.score(X_test, y_test) * 100
```

```
78.35929767120722
```

- **Modeling using Tensorflow – Neuron Network**

History Traning using Tenserflow



Model Summary

```
[51] model.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 100)               1200

 dense_1 (Dense)             (None, 100)               10100

 dense_2 (Dense)             (None, 1)                 101

=================================================================
Total params: 11,401
Trainable params: 11,401
Non-trainable params: 0
_________________________________________________________________
```

Referensi: