TUGAS PRAKTIKUM KECERDASAN BUATAN

Untuk Memenuhi Pertemuan-11

“Linear Regression dan Neural Network ”

Dosen Pengampu: Leni Fitriani, ST. M.Kom

Disusun Oleh :

Agil Rahmat (2106037)

Informatika A

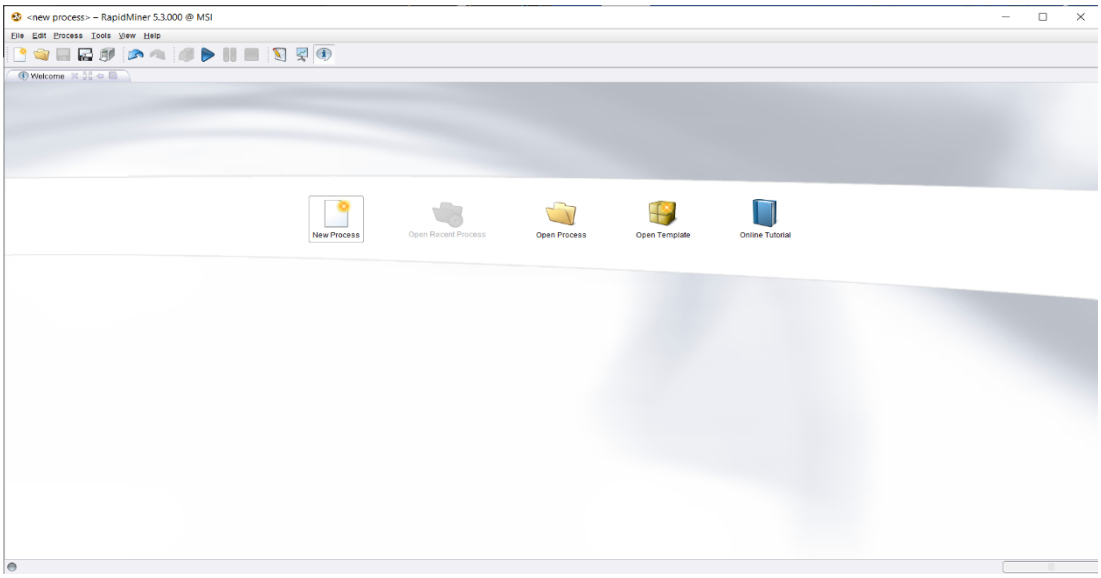
PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI GARUT

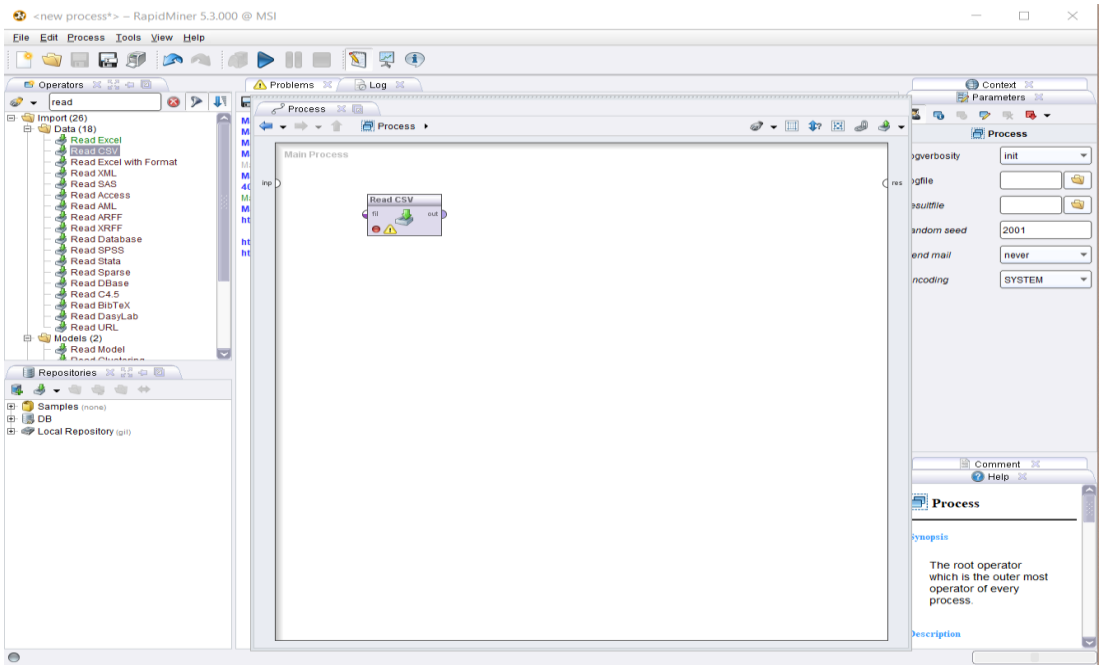
2023

1. Tahapan Rapid Miner Linear Regression.

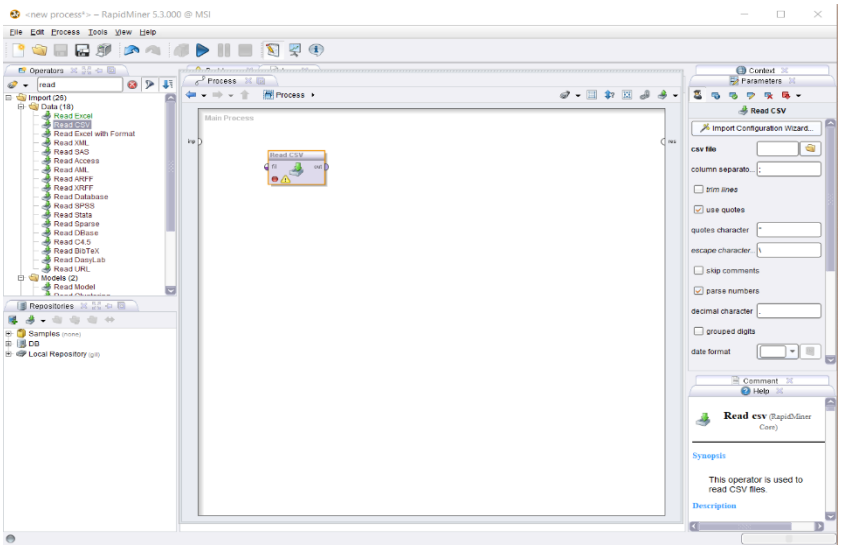
   1) Buka RapidMiner Studio dan buatlah sebuah proses baru. Pilih New Process.



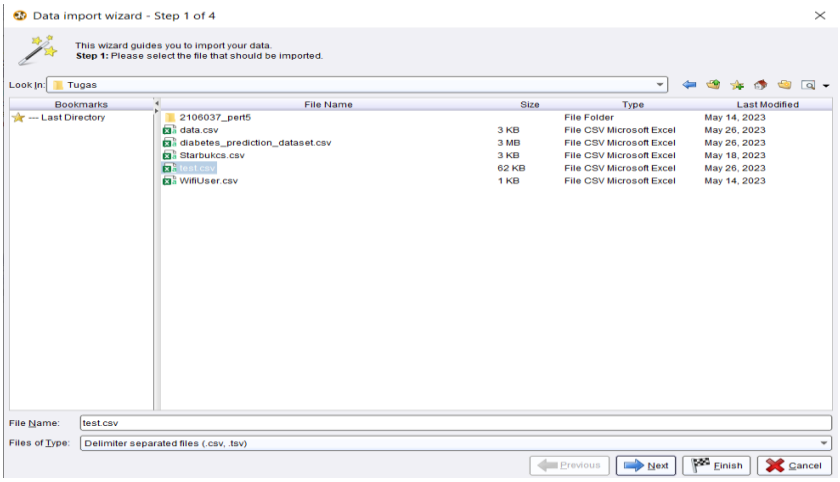
   2) Tambahkan Operator “Read CSV” dan masukkan kedalam bagian Process.



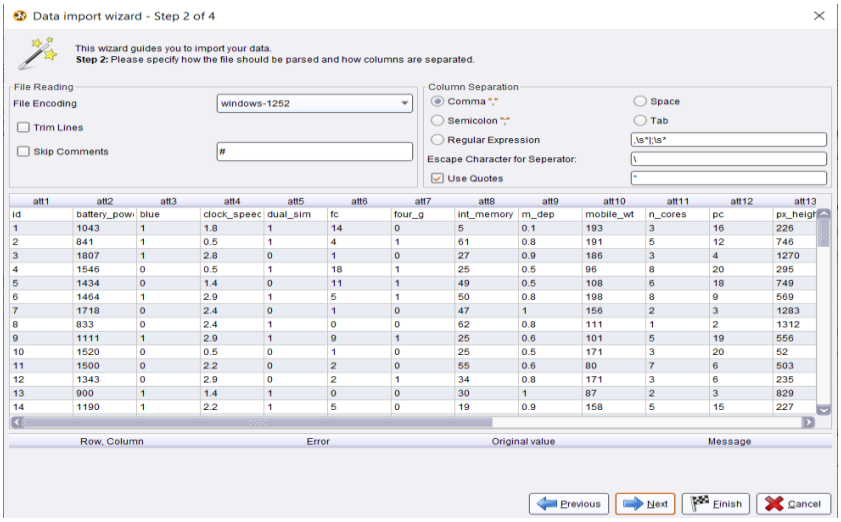
3) Klik Operator “Read CSV”, kemudian klik tombol “Import Configuration Wizard” pada bagian Parameter read csv tersebut.



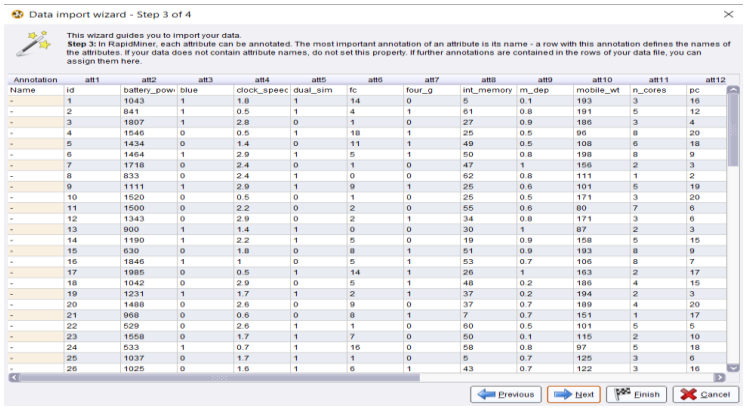
4) Pilih dataset yang hendak digunakan kemudian klik next.



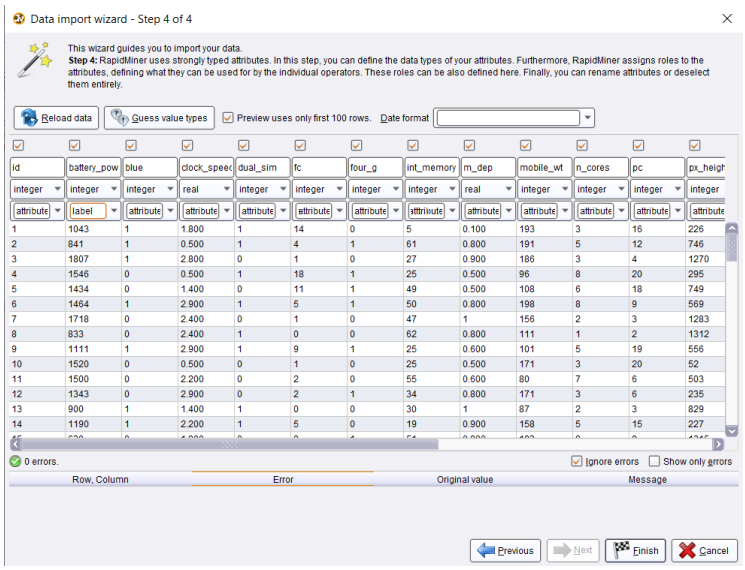
5) Tentukan comma sebagai column separator nya, kemudian klik next.



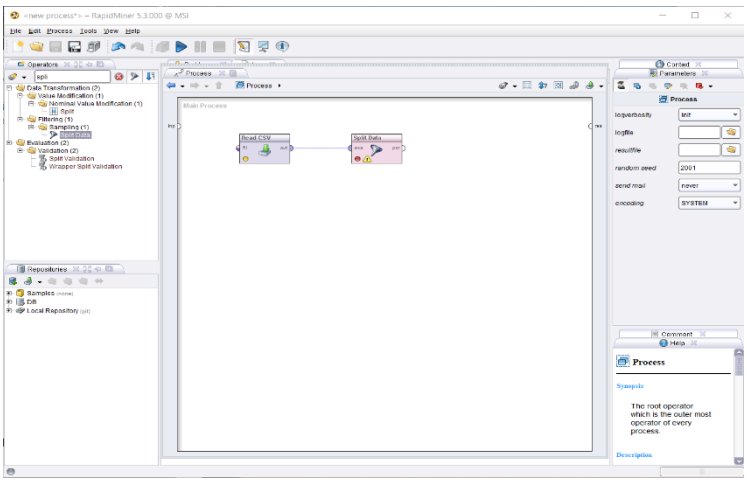
6) Klik next



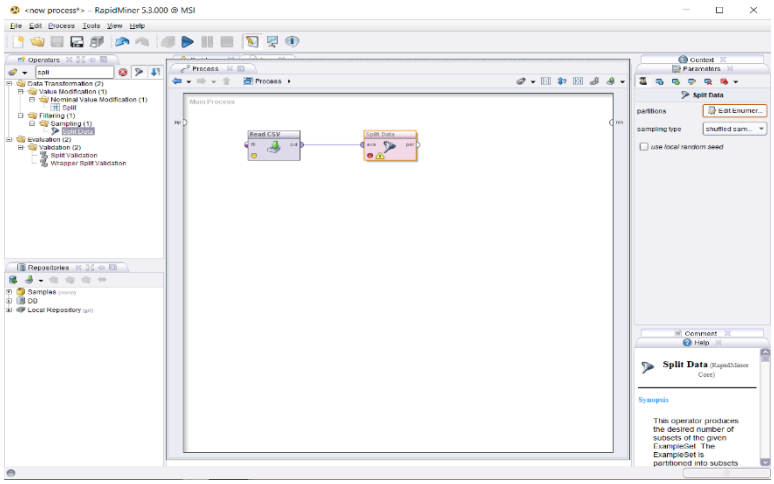
7) Atur Kolom “battery_power” dan ganti attribute nya menjadi “label”. Setelah itu klik finish.



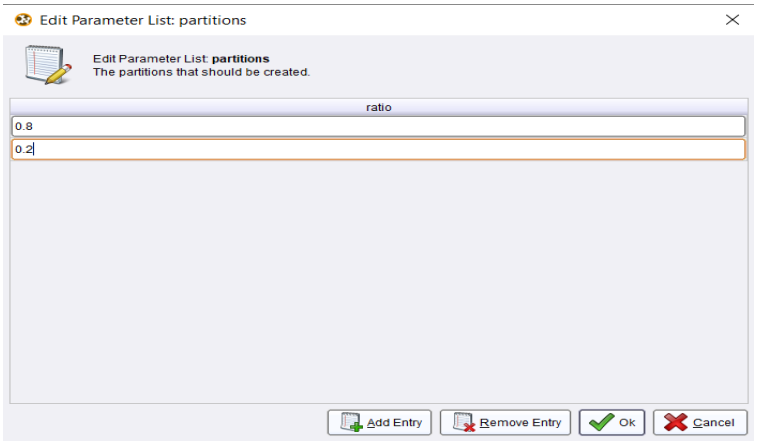
8) Masukkan Operator “Split Data” dan hubungkan dengan Operator “Read CSV”.



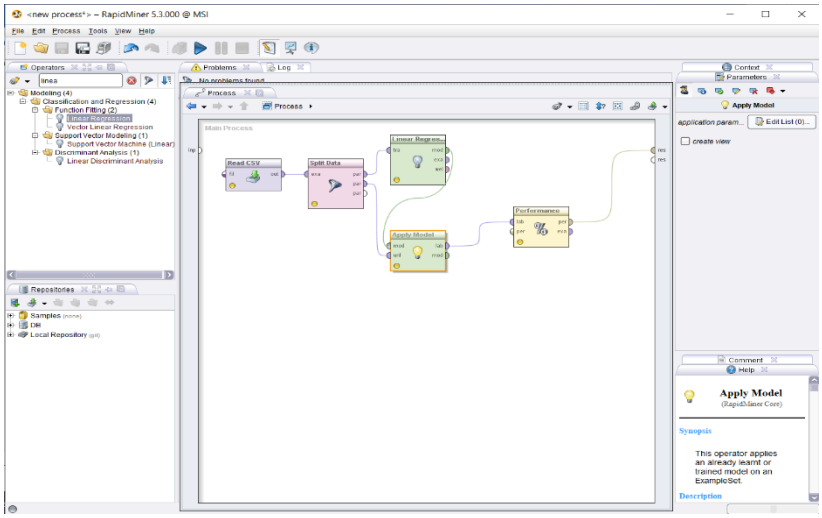
9) Klik Operator “Split Data” kemudian klik “Edit Enumeration” pada bagian Parameter split data tersebut.



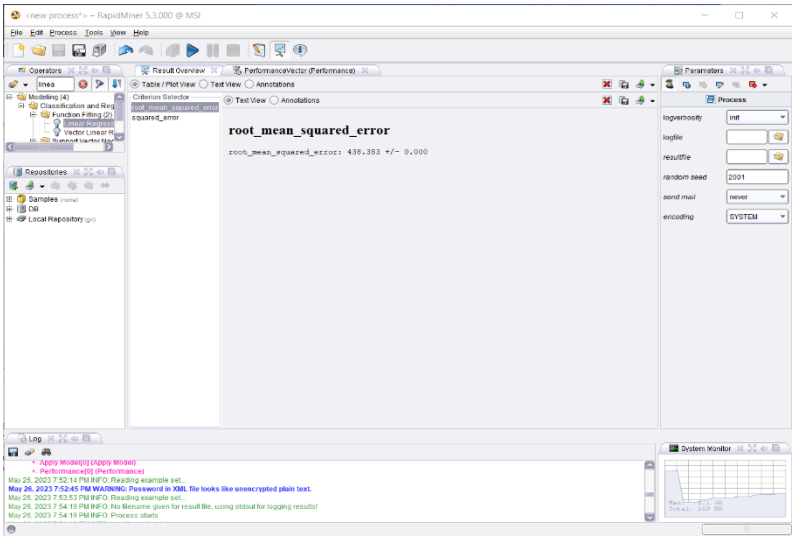
10) Split data menjadi 80% untuk training (0.8) dan 20% untuk testing (0.2). Jika sudah klik OK.



11) Masukkan Operator “Linear Regression”, “Apply Model”, dan“Performance(Regression)” kemudian hubungkan seperti gambar dibawah ini:

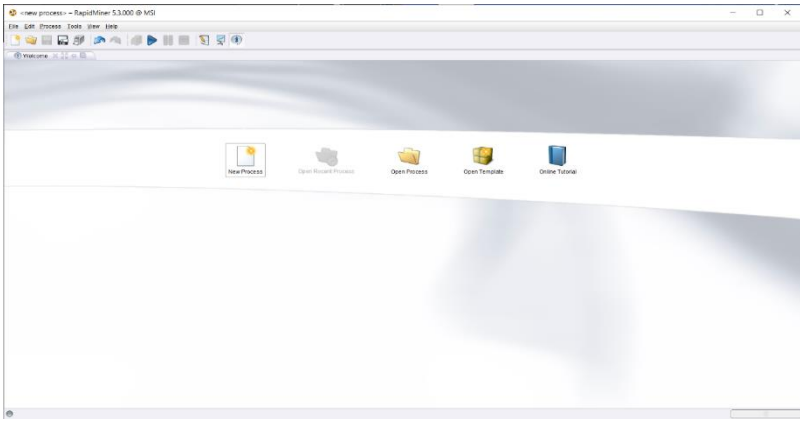


12) Run atau jalankan untuk melihat hasilnya.

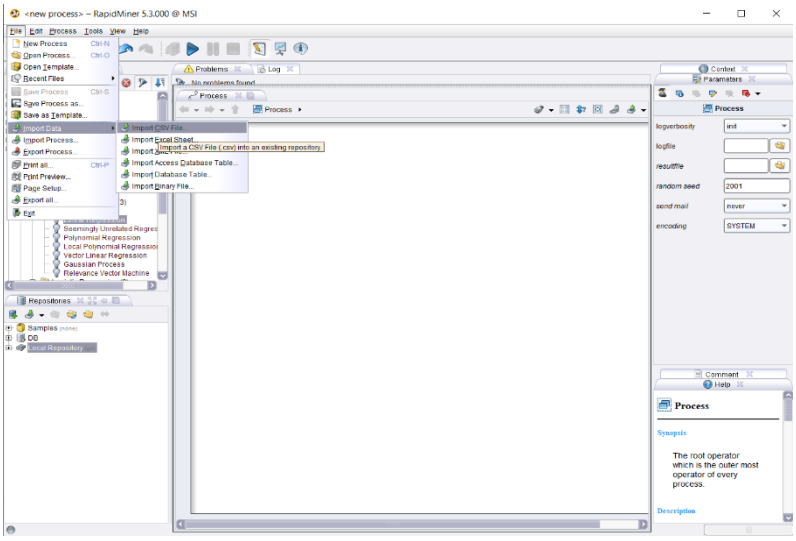


2. Tahapan RapidMiner Neutral Network

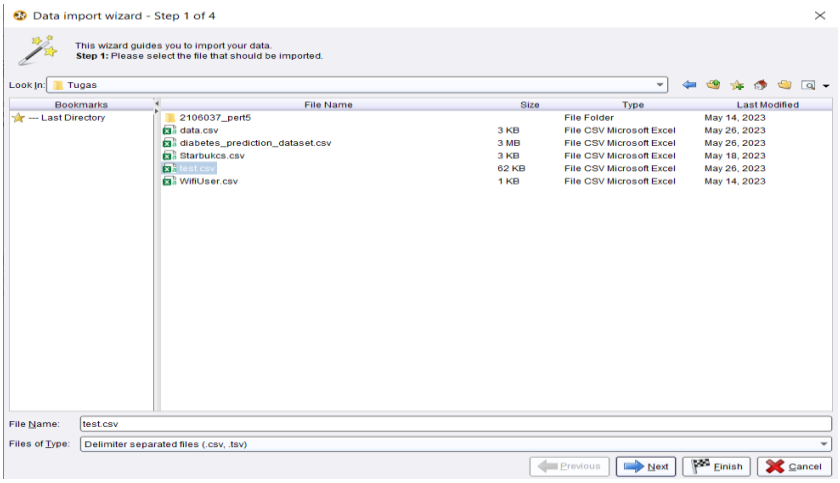
1) Klik New Process.



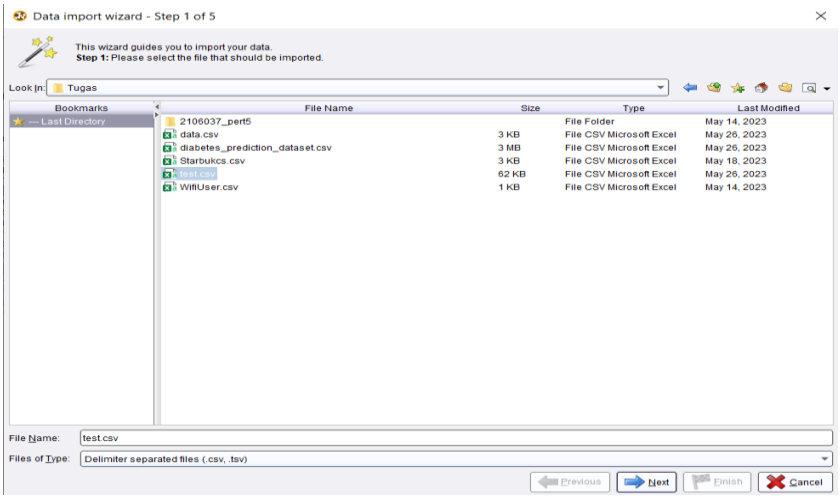
2) Klik menu File, lalu klik sub menu Import Data disitu ada beberapa pilihan untuk mengimport data berdasarkan ekstensi dari data. Pilih Import CSV Sheet



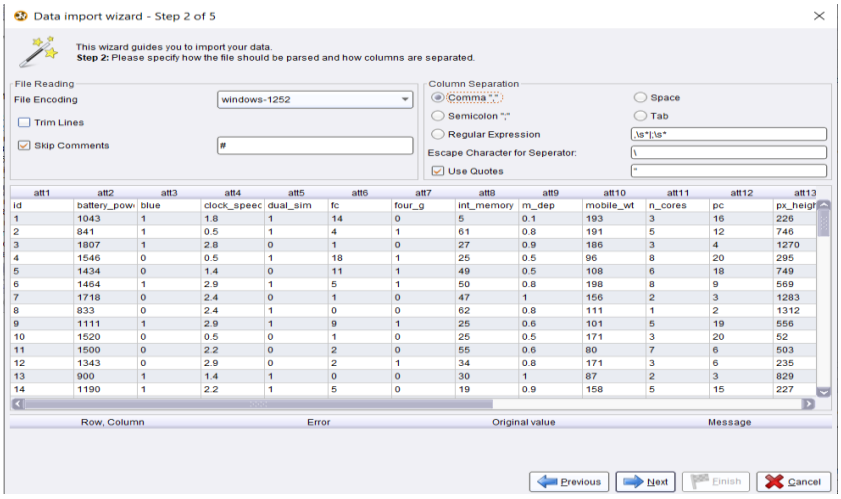
3) Pilih dataset yang hendak digunakan kemudian klik next.



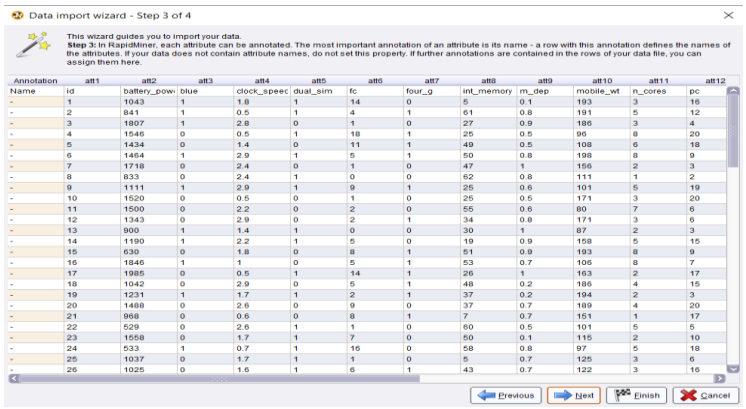
4) Tentukan comma sebagai column separator nya, kemudian klik next.



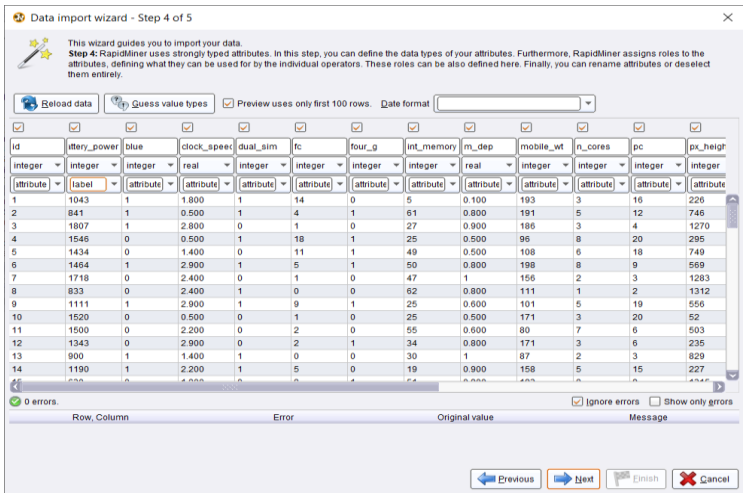
5) Tentukan comma sebagai column separator nya, kemudian klik next.



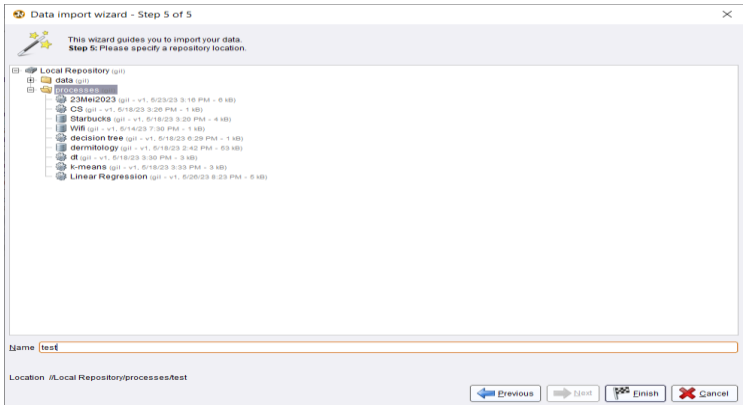
6) Klik next



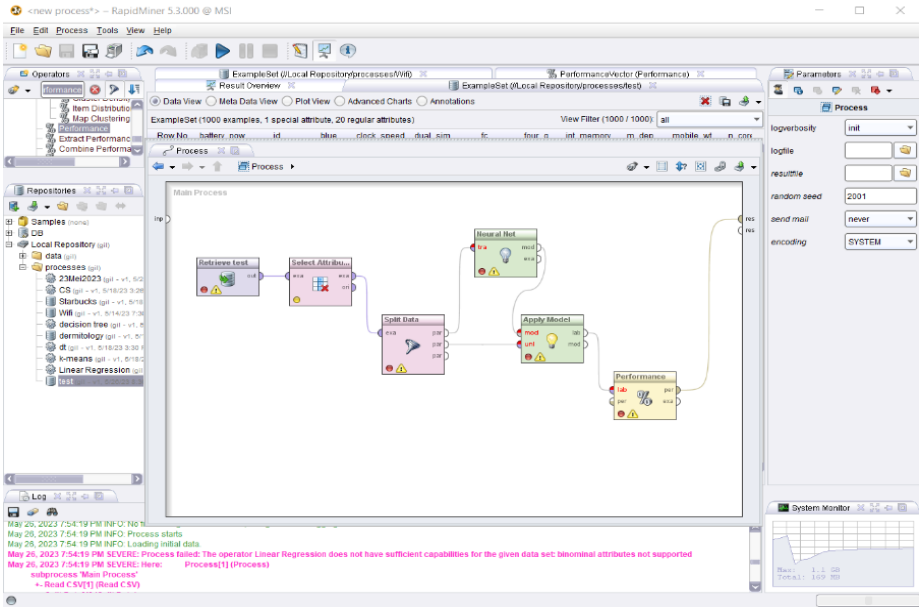
7) Atur Kolom “battery_power” dan ganti attribute nya menjadi “label”. Setelah itu klik next.



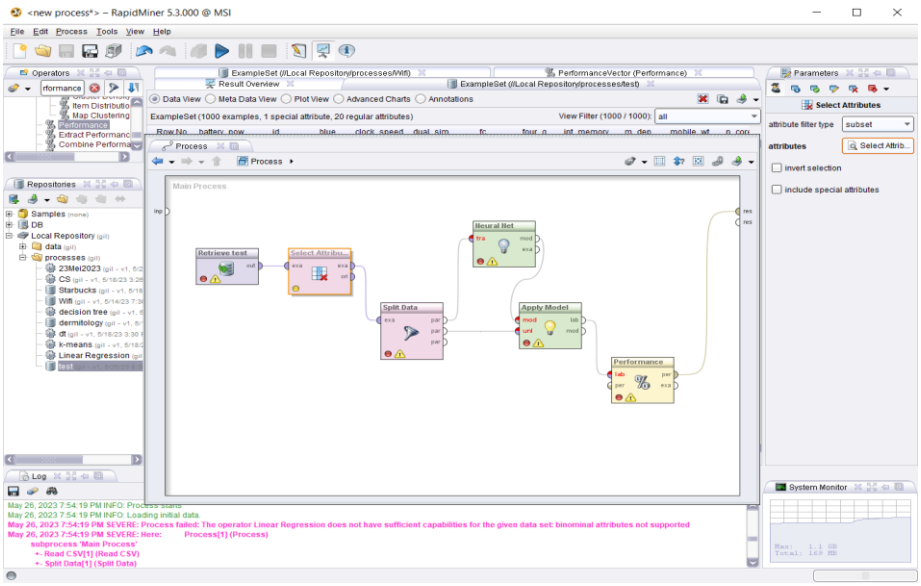
8) Tentukan tempat penyimpanannya, berinama juga untuk file nya dan terakhir bisa klik finisih.



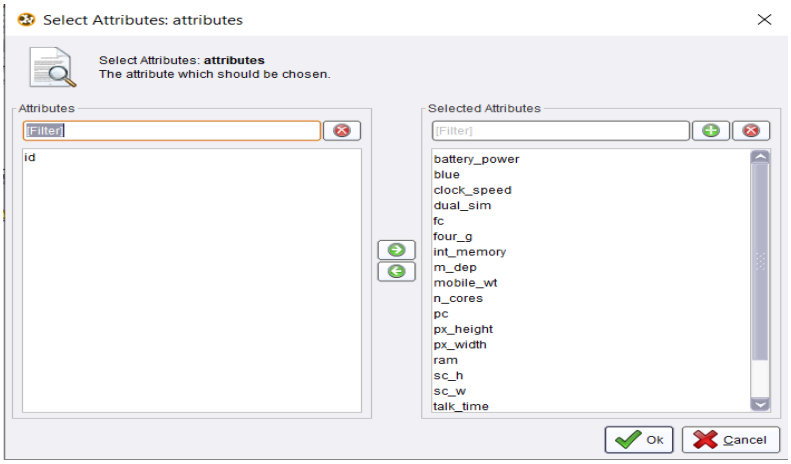
9) Masukkan dataset test dari penyimpanan data tadi, lalu masukkan juga Operator “Select Attributes”, “Split Data”, “Neural Net”, “Apply Model”, dan “Performance (Classification). Kemudian hubungkan seperti gambar dibawah ini:



10) Klik Operator “Select Attributes”, kemudian pada Parameter select attributes tersebut terdapat “attribute filter type” ubah lah menjadi “subset” dan lalu juga klik dibawahnya yaitu “Select Attributes”.

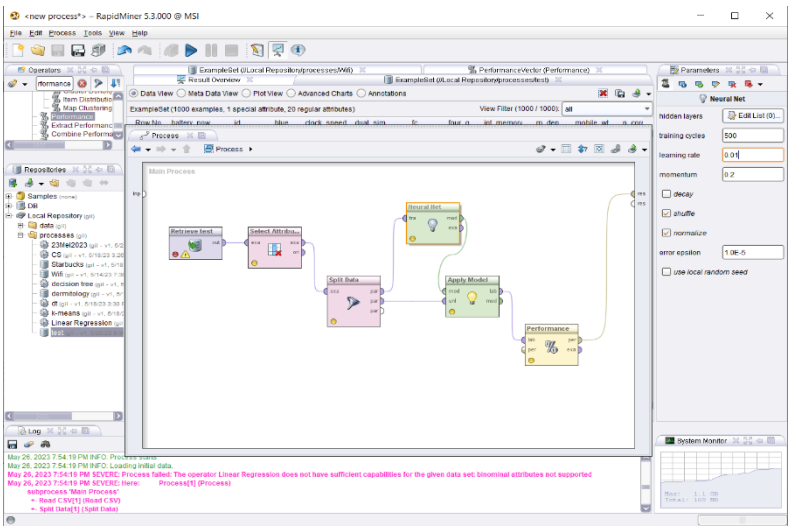


11) Tentukan atribut mana saja yang akan digunakan (disini kita akan mengecualikan atribut id). Kemudian klik OK.

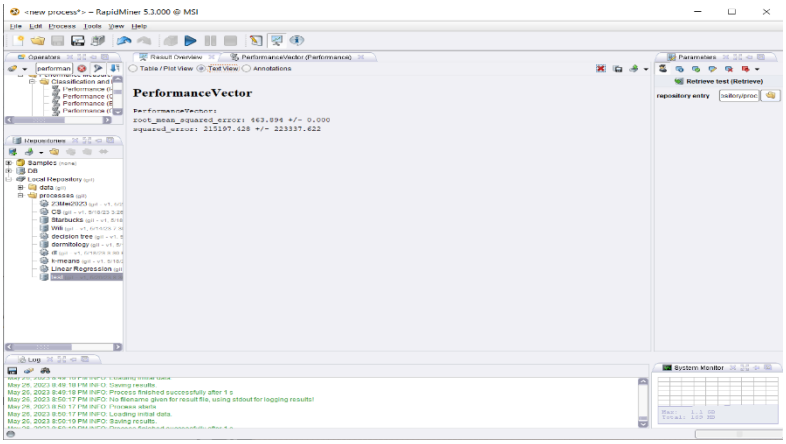


12) Klik Operator “Split Data” kemudian klik “Edit Enumeration” pada bagian Parameter split data tersebut. Lalu ubah Split data menjadi 80% untuk training (0.8) dan 20% untuk testing (0.2). Jika sudah klik OK.

13) Klik Operator “Neural Net” kemudian atur “Learning Rate” pada Parameter neural net tersebut menjadi 0.01.



14) Run atau Jalankan untuk melihat hasilnya.



3. Python (Linear Regression)

1) Load Library

```
[ ]  import pandas as pd
     import matplotlib.pyplot as plt
     from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import train_test_split
```

2) Load Datasets

```
[ ]  df = pd.read_csv('test.csv', usecols=['battery_power','clock_speed'])
```

3) Sneak Peak Data

```
df.head()
```

	battery_power	clock_speed
0	1043	1.8
1	841	0.5
2	1807	2.8
3	1546	0.5
4	1434	1.4

```
[ ]  df.shape
```

```
(1000, 2)
```

```
[ ] df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 2 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  1000 non-null   int64
 1   clock_speed    1000 non-null   float64
dtypes: float64(1), int64(1)
memory usage: 15.8 KB
```

```
df.describe()
```

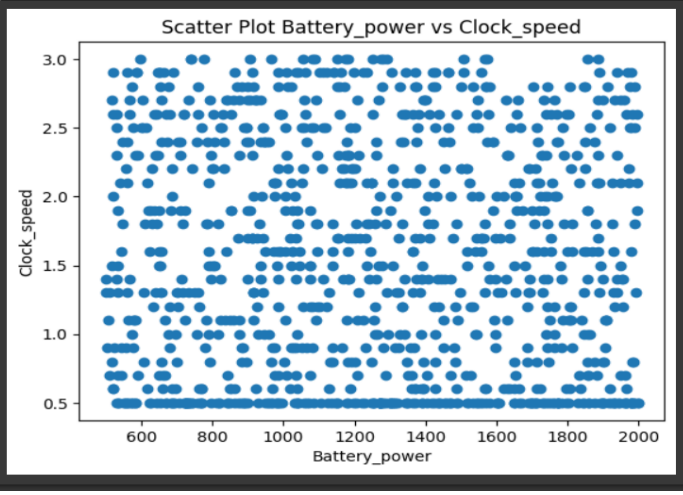
	battery_power	clock_speed
count	1000.000000	1000.000000
mean	1248.510000	1.540900
std	432.458227	0.829268
min	500.000000	0.500000
25%	895.000000	0.700000
50%	1246.500000	1.500000
75%	1629.250000	2.300000
max	1999.000000	3.000000

4) Handling Missing Values

```
[ ] df.isnull().sum()
```

```
battery_power    0
clock_speed      0
dtype: int64
```

```
plt.scatter(df['battery_power'], df['clock_speed'])
plt.xlabel('Battery_power')
plt.ylabel('Clock_speed')
plt.title('Scatter Plot Battery_power vs Clock_speed')
plt.show()
```



```
df.corr()
```

	battery_power	clock_speed
battery_power	1.000000	-0.039075
clock_speed	-0.039075	1.000000

5) Modelling

```
[17] df.head()
```

	battery_power	clock_speed
0	1043	1.8
1	841	0.5
2	1807	2.8
3	1546	0.5
4	1434	1.4

```
[18] x = df['battery_power'].values.reshape(-1,1)
     y = df['clock_speed'].values.reshape(-1,1)

     x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2)

[20] lin_reg = LinearRegression()

[21] lin_reg.fit(x_train, y_train)
```

```
LinearRegression
LinearRegression()
```

```
print(lin_reg.coef_)
print(lin_reg.intercept_)
```

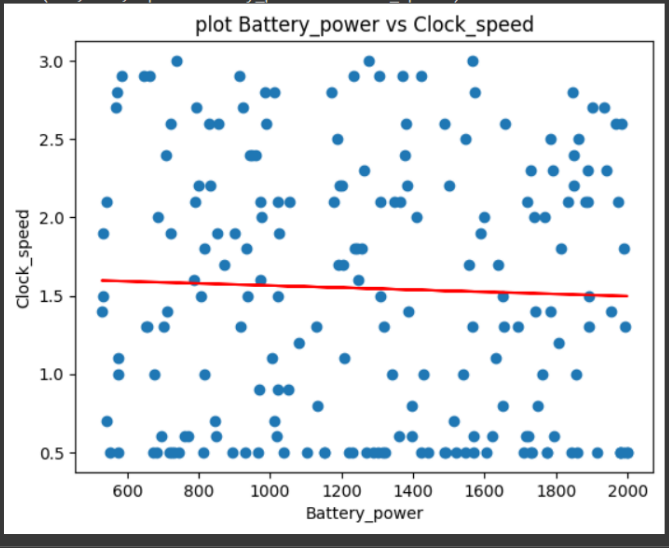
```
[[-6.75726378e-05]]
[1.6321296]
```

```
[23] lin_reg.score(x_test, y_test)
```

```
0.0007209293767724834
```

```
y_prediksi = lin_reg.predict(x_test)
plt.scatter(x_test, y_test)
plt.plot(x_test, y_prediksi, c='r')
plt.xlabel('Battery_power')
plt.ylabel('Clock_speed')
plt.title('plot Battery_power vs Clock_speed')
```

```
Text(0.5, 1.0, 'plot Battery_power vs Clock_speed')
```



6) Prediction

```
[25] lin_reg.predict([[100]])

     array([[1.62537234]])

[26] lin_reg.predict([[150]])

     array([[1.62199371]])

     lin_reg.predict([[200]])

     array([[1.61861508]])
```

4. Python (Neural Network)

1) Load Libraries

```
[1]  import keras
     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     import numpy as np
     import pandas as pd
     from sklearn.preprocessing import normalize
```

2) Reading Data

```
data = pd.read_csv("test.csv")
print("Describing the data: ",data.describe())
print("Info of the data: ",data.info())
```

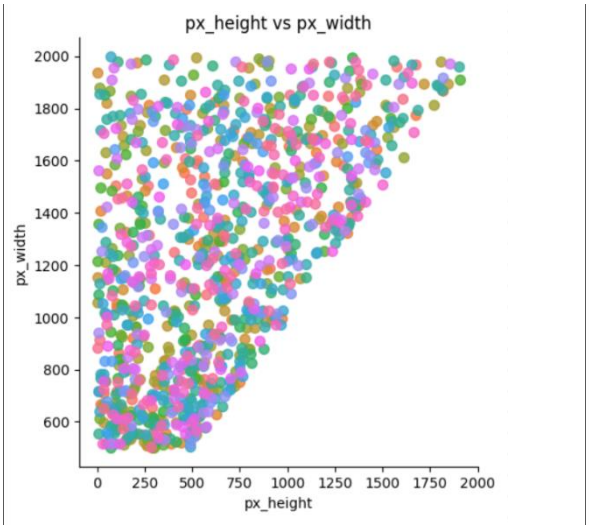
```
Describing the data:                 id  battery_power         blue  clock_speed     dual_sim  \
count  1000.000000    1000.000000  1000.000000  1000.000000  1000.000000
mean    500.500000    1248.510000     0.516000     1.540900     0.517000
std     288.819436     432.458227     0.499994     0.829268     0.499961
min       1.000000     500.000000     0.000000     0.500000     0.000000
25%     250.750000     895.000000     0.000000     0.700000     0.000000
50%     500.500000    1246.500000     1.000000     1.500000     1.000000
75%     750.250000    1629.250000     1.000000     2.300000     1.000000
max    1000.000000    1999.000000     1.000000     3.000000     1.000000

                fc       four_g   int_memory        m_dep   mobile_wt  ...  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.00000  ...
mean      4.593000     0.487000    33.652000     0.517500   139.51100  ...
std       4.463325     0.500081    18.128694     0.280861    34.85155  ...
min       0.000000     0.000000     2.000000     0.100000    80.00000  ...
25%       1.000000     0.000000    18.000000     0.300000   109.75000  ...
50%       3.000000     0.000000    34.500000     0.500000   139.00000  ...
75%       7.000000     1.000000    49.000000     0.800000   170.00000  ...
max      19.000000     1.000000    64.000000     1.000000   200.00000  ...

                pc    px_height     px_width          ram         sc_h  \
count  1000.000000  1000.000000  1000.000000  1000.000000  1000.000000
mean     10.054000   627.121000  1239.774000  2138.998000    11.995000
std       6.095099   432.929699   439.670981  1088.092278     4.320607
min       0.000000     0.000000   501.000000   263.000000     5.000000
25%       5.000000   263.750000   831.750000  1237.250000     8.000000
50%      10.000000   564.500000  1250.000000  2153.500000    12.000000
75%      16.000000   903.000000  1637.750000  3065.500000    16.000000
max      20.000000  1907.000000  1998.000000  3989.000000    19.000000

               sc_w    talk_time      three_g  touch_screen         wifi
count  1000.000000  1000.000000  1000.000000    1000.00000  1000.000000
mean      5.316000    11.085000     0.756000       0.50000     0.507000
std       4.240062     5.497636     0.429708       0.50025     0.500201
min       0.000000     2.000000     0.000000       0.00000     0.000000
25%       2.000000     6.750000     1.000000       0.00000     0.000000
50%       5.000000    11.000000     1.000000       0.50000     1.000000
75%       8.000000    16.000000     1.000000       1.00000     1.000000
max      18.000000    20.000000     1.000000       1.00000     1.000000

[8 rows x 21 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             1000 non-null   int64
 1   battery_power  1000 non-null   int64
 2   blue           1000 non-null   int64
 3   clock_speed    1000 non-null   float64
 4   dual_sim       1000 non-null   int64
 5   fc             1000 non-null   int64
 6   four_g         1000 non-null   int64
 7   int_memory     1000 non-null   int64
 8   m_dep          1000 non-null   float64
 9   mobile_wt      1000 non-null   int64
 10  n_cores        1000 non-null   int64
 11  pc             1000 non-null   int64
 12  px_height      1000 non-null   int64
 13  px_width       1000 non-null   int64
 14  ram            1000 non-null   int64
 15  sc_h           1000 non-null   int64
 16  sc_w           1000 non-null   int64
 17  talk_time      1000 non-null   int64
 18  three_g        1000 non-null   int64
 19  touch_screen   1000 non-null   int64
 20  wifi           1000 non-null   int64
dtypes: float64(2), int64(19)
memory usage: 164.2 KB
Info of the data:  None
```

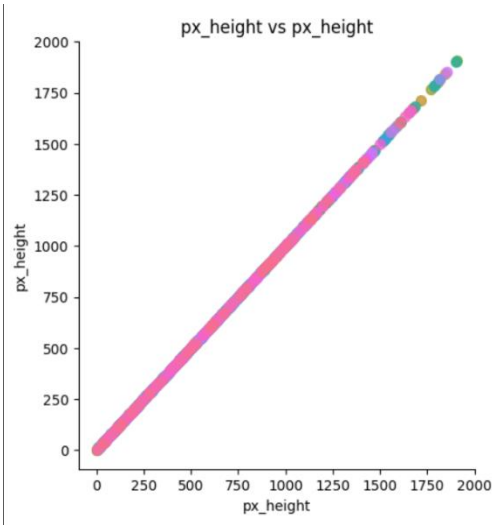
```
print("10 first samples of the dataset: ",data.head(10))
print("10 last samples of the dataset: ",data.tail(10))
```

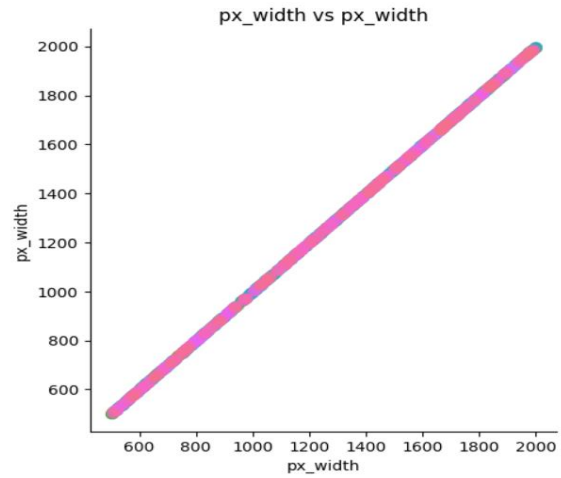
```
10 first samples of the dataset:     id  battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
0   1           1043     1          1.8         1  14       0           5
1   2            841     1          0.5         1   4       1          61
2   3           1807     1          2.8         0   1       0          27
3   4           1546     0          0.5         1  18       1          25
4   5           1434     0          1.4         0  11       1          49
5   6           1464     1          2.9         1   5       1          50
6   7           1718     0          2.4         0   1       0          47
7   8            833     0          2.4         1   0       0          62
8   9           1111     1          2.9         1   9       1          25
9  10           1520     0          0.5         0   1       0          25

   m_dep  mobile_wt  ...  pc  px_height  px_width   ram  sc_h  sc_w  \
0    0.1        193  ...  16        226      1412  3476    12     7
1    0.8        191  ...  12        746       857  3895     6     0
2    0.9        186  ...   4       1270      1366  2396    17    10
3    0.5         96  ...  20        295      1752  3893    10     0
4    0.5        108  ...  18        749       810  1773    15     8
5    0.8        198  ...   9        569       939  3506    10     7
6    1.0        156  ...   3       1283      1374  3873    14     2
7    0.8        111  ...   2       1312      1880  1495     7     2
8    0.6        101  ...  19        556       876  3485    11     9
9    0.5        171  ...  20         52      1009   651     6     0

   talk_time  three_g  touch_screen  wifi
0          2        0             1     0
1          7        1             0     0
2         10        0             1     1
3          7        1             1     0
4          7        1             0     1
5          3        1             1     1
```

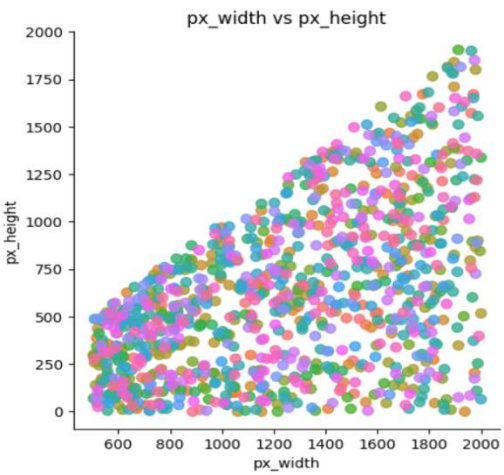
3) Visualisation of the dataset

```
[16] sns.lmplot(x='px_height',y='px_width',
                data=data,
                fit_reg=False,
                hue="battery_power",
                scatter_kws={"marker":"D",
                             "s":50})
     plt.title('px_height vs px_width')

     sns.lmplot(x='px_height', y='px_height',
                data=data,
                fit_reg=False,
                hue="battery_power",
                scatter_kws={"marker":"D",
                             "s":50})
     plt.title('px_height vs px_height')

     sns.lmplot(x='px_width', y='px_width',
                data=data,
                fit_reg=False,
                hue="battery_power",
                scatter_kws={"marker":"D",
                             "s":50})
     plt.title('px_width vs px_width')

     sns.lmplot(x='px_width', y='px_height',
                data=data,
                fit_reg=False,
                hue="battery_power",
                scatter_kws={"marker":"D",
                             "s":50})
     plt.title('px_width vs px_height')
     plt.show()
```









Continue:

```
print(data["battery_power"].unique())
```

```
[1043  841 1807 1546 1434 1464 1718  833 1111 1520 1500 1343  900 1190
 630 1846 1985 1042 1231 1488  968  529 1558  533 1037 1025 1858  980
 644 1024 1981 1380 1557 1201 1074 1175 1280 1715 1165  567 1952  822
 685 1388 1972 1411 1094 1653  916 1712  882  632 1442 1630 1596 1272
1640 1889 1907  578 1634 1533  660 1847 1206  549 1705 1366 1991 1102
1452 1810 1166  881 1134 1031 1376 1391  979 1075 1999 1626  942 1182
1982 1373 1151 1650 1663 1965  679 1465 1809  757 1034 1119  559 1204
1008 1397  697 1939 1039 1605  769  861  504 1930 1795 1363 1901 1319
 859 1664  955  517 1806 1348 1455 1611 1573  557 1599 1051 1857 1986
 591 1140  923 1582  723 1251  574  948 1571  564 1466  597  895 1535
1832 1045 1483  976 1840  624 1963 1307 1933 1496 1532 1004  945 1081
1012 1762  796 1547  988 1180  852  607 1765 1250 1577 1153  651 1186
1429  556 1735 1859  915  890  758  541  586  762  683 1526 1771  639
1783 1384 1770 1202  885 1629 1072 1863 1739 1278  562 1249 1811  560
1773  725  800 1300 1001  500 1378  951 1038  987 1698  785 1851 1976
1649 1157 1702 1040  790  739 1364 1580 1519  989 1240 1273 1784 1169
 703  959 1292 1927 1477 1759  831 1803 1361 1183  917 1247  930 1019
1855 1730 1887 1590  603 1396 1598 1692  628 1883  819 1367 1744 1086
 899  896 1714 1395  786 1687  669 1423 1600 1621 1672 1934 1700 1550
 626 1737 1370 1899  848 1430  768 1856 1729 1210 1880  996 1263 1956
 992  911 1979 1095  656 1394 1632 1225 1337 1178 1269  986 1904  803
1399  788  579 1010  530 1289 1877 1764 1709 1690  600  743 1850 1297
1706 1967 1392 1248  894 1829 1372  706  649 1137 1320  792  918  929
 767 1315  569  712 1603 1472 1543  652  546 1408 1021 1658 1088  542
 576  950  518 1996 1197 1006 1679 1262  904 1834  681  687  863  944
1283  817  804 1177 1997 1745 1187 1422  658  521  666  805 1655 1224
1234 1093  583 1988 1746  907 1427 1101 1403  657 1688  876 1065 1235
1959  650 1030  853  744  690  590 1266  815 1893  717 1439  983  718
 776 1018  875 1341 1719 1703  701 1560  695 1487  511  507 1490 1257
 926 1303 1812 1943 1082  721  654 1948 1242 1453 1232  795 1915  532
1767 1667 1346  958 1176 1911 1789  972 1256 1900  756 1728 1970  888
1419  588 1607 1171 1138 1096  906  740 1435 1995 1813  812  977 1125
1733 1333  964 1613 1123 1913 1302  797 1502 1457 1514 1604 1919 1152
```

```
data.loc[data["battery_power"]=="1949","battery_power"]=0
data.loc[data["battery_power"]=="1556","battery_power"]=1
data.loc[data["battery_power"]=="1324","battery_power"]=2

print(data.head())
```

```
      id  battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
102  103           1008     0          2.3         1   4       1          15
149  150           1483     1          0.8         0   4       1          61
818  819           1247     0          0.8         1  16       0          24
889  890           1162     1          2.9         1   2       0          57
20    21            968     0          0.6         0   8       1           7

     m_dep  mobile_wt  ...  pc  px_height  px_width   ram  sc_h  sc_w  \
102    0.4         89  ...  19        491       692   450    11     7
149    0.6        128  ...  10        655       814  3843    11     8
818    0.8        116  ...  20       1114      1350   522     9     2
889    0.3        146  ...  19          9       555  1316    10     2
20     0.7        151  ...  17        504      1930  1357    15     1

     talk_time  three_g  touch_screen  wifi
102         18        1             0     0
149         20        1             0     1
818         20        0             1     1
889          3        1             0     1
20          16        1             1     0

[5 rows x 21 columns]
```

```
data=data.iloc[np.random.permutation(len(data))]
print(data.head())
```

```
      id  battery_power  blue  clock_speed  dual_sim  fc  four_g  int_memory  \
733  734           1651     0          0.5         0   1       1          57
587  588            763     0          1.2         1   4       1          59
826  827           1036     0          1.4         1   6       1          37
121  122            955     0          0.5         1   0       0          62
304  305           1210     1          2.3         0   0       0          63

     m_dep  mobile_wt  ...  pc  px_height  px_width   ram  sc_h  sc_w  \
733    0.2        193  ...   6        697       924  3421     9     0
587    0.6        119  ...  17        724       737  2863    18     3
826    0.5        184  ...  10        416       523  3864     5     2
121    0.4        151  ...   1        261       538  3758    15     3
304    0.9        141  ...   4         51      1761   977    14     6

     talk_time  three_g  touch_screen  wifi
733         15        1             1     1
587         17        1             0     1
826         13        1             1     1
121         18        1             1     1
304         20        1             0     0

[5 rows x 21 columns]
```

```
X=data.iloc[:,1:5].values
y=data.iloc[:,5].values

print("Shape of X",X.shape)
print("Shape of y",y.shape)
print("Examples of X\n",X[:3])
print("Examples of y\n",y[:3])
```

```
Shape of X (1000, 4)
Shape of y (1000,)
Examples of X
 [[1.008e+03 0.000e+00 2.300e+00 1.000e+00]
 [1.483e+03 1.000e+00 8.000e-01 0.000e+00]
 [1.247e+03 0.000e+00 8.000e-01 1.000e+00]]
Examples of y
 [ 4  4 16]
```

4) Normalization

```
X_normalized=normalize(X,axis=0)
print("Examples of X_normalised\n",X_normalized[:3])
```

```
Examples of X_normalised
 [[0.02412609 0.         0.04156903 0.04397995]
 [0.03549503 0.04402255 0.01445879 0.        ]
 [0.02984646 0.         0.01445879 0.04397995]]
```

```
'''
80% -- train data
20% -- test data
'''
total_length=len(data)
train_length=int(0.8*total_length)
test_length=int(0.2*total_length)

X_train=X_normalized[:train_length]
X_test=X_normalized[train_length:]
y_train=y[:train_length]
y_test=y[train_length:]

print("Length of train set x:",X_train.shape[0],"y:",y_train.shape[0])
print("Length of test set x:",X_test.shape[0],"y:",y_test.shape[0])
```

```
Length of train set x: 800 y: 800
Length of test set x: 200 y: 200
```

```
[72] from keras.models import Sequential
     from keras.layers import Dense,Activation,Dropout
     from keras.utils import np_utils
```

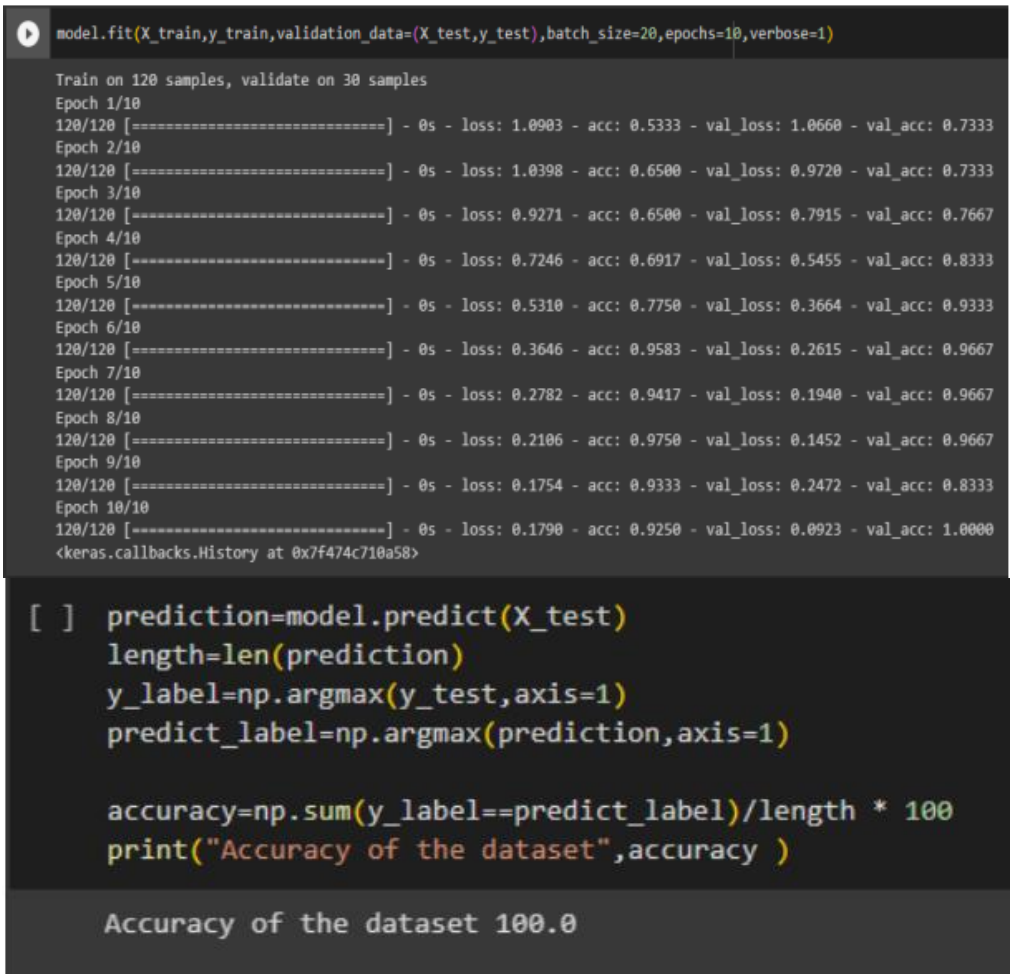
```
[86] '''
     [0]--->[1 0 0]
     [1]--->[0 1 0]
     [2]--->[0 0 1]
     '''
     X_train=np_utils.to_categorical(X_train,num_classes=3)
     X_test=np_utils.to_categorical(X_test,num_classes=3)
     print("Shape of X_tarin",y_train.shape)
     print("Shape of X_test",y_test.shape)
```

```
Shape of X_tarin (800, 4, 3, 3, 3, 3)
Shape of X_test (200,)
```

```
model=Sequential()
model.add(Dense(1000,input_dim=4,activation='relu'))
model.add(Dense(500,activation='relu'))
model.add(Dense(300,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(3,activation='relu'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_8 (Dense)             (None, 1000)              5000

 dense_9 (Dense)             (None, 500)               500500

 dense_10 (Dense)            (None, 300)               150300

 dropout_2 (Dropout)         (None, 300)               0

 dense_11 (Dense)            (None, 3)                 903

=================================================================
Total params: 656,703
Trainable params: 656,703
Non-trainable params: 0
_________________________________________________________________
```



```
model.fit(X_train,y_train,validation_data=(X_test,y_test),batch_size=20,epochs=10,verbose=1)

Train on 120 samples, validate on 30 samples
Epoch 1/10
120/120 [==============================] - 0s - loss: 1.0903 - acc: 0.5333 - val_loss: 1.0660 - val_acc: 0.7333
Epoch 2/10
120/120 [==============================] - 0s - loss: 1.0398 - acc: 0.6500 - val_loss: 0.9720 - val_acc: 0.7333
Epoch 3/10
120/120 [==============================] - 0s - loss: 0.9271 - acc: 0.6500 - val_loss: 0.7915 - val_acc: 0.7667
Epoch 4/10
120/120 [==============================] - 0s - loss: 0.7246 - acc: 0.6917 - val_loss: 0.5455 - val_acc: 0.8333
Epoch 5/10
120/120 [==============================] - 0s - loss: 0.5310 - acc: 0.7750 - val_loss: 0.3664 - val_acc: 0.9333
Epoch 6/10
120/120 [==============================] - 0s - loss: 0.3646 - acc: 0.9583 - val_loss: 0.2615 - val_acc: 0.9667
Epoch 7/10
120/120 [==============================] - 0s - loss: 0.2782 - acc: 0.9417 - val_loss: 0.1940 - val_acc: 0.9667
Epoch 8/10
120/120 [==============================] - 0s - loss: 0.2106 - acc: 0.9750 - val_loss: 0.1452 - val_acc: 0.9667
Epoch 9/10
120/120 [==============================] - 0s - loss: 0.1754 - acc: 0.9333 - val_loss: 0.2472 - val_acc: 0.8333
Epoch 10/10
120/120 [==============================] - 0s - loss: 0.1790 - acc: 0.9250 - val_loss: 0.0923 - val_acc: 1.0000
<keras.callbacks.History at 0x7f474c710a58>

[ ]  prediction=model.predict(X_test)
     length=len(prediction)
     y_label=np.argmax(y_test,axis=1)
     predict_label=np.argmax(prediction,axis=1)

     accuracy=np.sum(y_label==predict_label)/length * 100
     print("Accuracy of the dataset",accuracy )

     Accuracy of the dataset 100.0
```

Dalam dataset ‘test’ ini, tercapai akurasi 100%. Dapat dikatakan bahwa pada setiap epoch, jaringan saraf mencoba belajar dari fitur yang ada dan memprediksi berdasarkan bobot dan biasnya. Pada setiap epoch, bobot dan bias diubah dengan mengurangi tingkatannya untuk mencapai akurasi yang lebih baik setiap kali