```
 Iris_DataSet_ML_Models -1. SV - [1.1 Classification
(1ANN2LR3DTC4RFC5KNNC6SVM7NB)_1.2 Regression(1ANN2S-M-P-
LR3DTR4RFR5KNNR6SVM)] 2. USL -[K-C & HC]]
```

https://playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=circle&regDatase
plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=7,2&seed=0.10940&s
(https://playground.tensorflow.org/#activation=sigmoid&batchSize=10&dataset=circle&regDatase
plane&learningRate=0.03&regularizationRate=0&noise=0&networkShape=7,2&seed=0.10940&s

Artificial Intelligence

Machine Learning

Supervised Learning

Classification - Artificial Neural Network (classification)

- Logistic Regression / classification
- Decision Tree Classifier
- Random Forest Classifier
- K-N Neighbor Classifier
- Support Vector Machine - Linear, Ploly, RBF(Redial Basis Function)
- Naive Bayes - MultinomialNB, BurnoulliNB, GoussianNB

Regression - Artificial Neural Network (Regression)

- Simple Linear Regression
- Multi- Linear Regression
- Decision Tree Regresor
- Random Forest Regressor
- K -N Neighbour Regressor
- Support Vector Machine Learning - Linear, polly, RBF - Redial Basis Function

Unsupervised Learning

- K-Mean Clustering
- Hierarchical Clustering

1. import Liberary

In [1]:
```
import seaborn as sns
import pandas as pd
import numpy as np
```

2. Read data set

In [2]:
```
iris=pd.read_csv('iris.csv')
```

3. Analysing data set

In [3]:
```
df=iris
df.head()
```

Out[3]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

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

Out[4]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

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

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
Id	1.000000	0.716676	-0.397729	0.882747	0.899759
SepalLengthCm	0.716676	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.397729	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.882747	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.899759	0.817954	-0.356544	0.962757	1.000000

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

Out[7]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [9]:
```
df.nunique()
```

Out[9]:
```
Id               150
SepalLengthCm     35
SepalWidthCm      23
PetalLengthCm     43
PetalWidthCm      22
Species            3
dtype: int64
```

a. Artificial Neural Network - CLASSIFICATION

Classification Models using Artificial Neural Networks (ANN) -- (Artificial Neural Network Algorithm - CLASSIFICATION Models)

1. ANN Binary Classification

2. ANN multiclass Classification

- articals
- - https://www.analyticsvidhya.com/blog/2021/07/demystifying-the-difference-between-multi-class-and-multi-label-classification-problem-statements-in-deep-learning/ (https://www.analyticsvidhya.com/blog/2021/07/demystifying-the-difference-between-multi-class-and-multi-label-classification-problem-statements-in-deep-learning/)
- - https://machinelearningmastery.com/multi-label-classification-with-deep-learning/ (https://machinelearningmastery.com/multi-label-classification-with-deep-learning/)

1 ANN Binary Classification

- here should be two binary class in categorical column that will used in y (output, targrt, label & depentent variable)

- two class in y features

In [10]:
```
df.head(2),df.tail(2),df['Species'].nunique(), df['Species'].value_counts(),d
```

Out[10]:
```
(   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Specie
s
 0   1            5.1           3.5            1.4           0.2  Iris-setos
a
 1   2            4.9           3.0            1.4           0.2  Iris-setos
a,
       Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
 148  149            6.2           3.4            5.4           2.3
 149  150            5.9           3.0            5.1           1.8

            Species
 148  Iris-virginica
 149  Iris-virginica  ,
 3,
 Iris-setosa        50
 Iris-versicolor    50
 Iris-virginica     50
 Name: Species, dtype: int64,
 (150, 6))
```

Did two class in y ((output, targrt, label & depentent variable)

- df=df[df['Species']!=2] # did two class in label
- df=df[df['Species']!='Iris-virginica']

In [11]:
```
df=df[df['Species']!='Iris-virginica'] # did two class in label
```

```
```

In [12]:
```
df.head(2),df.tail(2),df['Species'].nunique(), df['Species'].value_counts(),d
```

Out[12]:
```
(   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Specie
s
 0   1            5.1           3.5            1.4           0.2  Iris-setos
a
 1   2            4.9           3.0            1.4           0.2  Iris-setos
a,
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  \
 98   99            5.1           2.5            3.0           1.1
 99  100            5.7           2.8            4.1           1.3

            Species
 98  Iris-versicolor
 99  Iris-versicolor  ,
 2,
 Iris-setosa        50
 Iris-versicolor    50
 Name: Species, dtype: int64,
 (100, 6))
```

4. LABEL Encodinging

4.1. by replace Pandas function

df['Species']=df['Species'].replace({'Iris-setosa':0, 'Iris-versicolor':1, 'Iris-virginica':2})

4.2. Label Encoder

- from sklearn.preprocessing import LabelEncoder
- le = LabelEncoder()
- df['Species'] = le.fit_transform(df['Species'])
- df

In [13]:
```
df['Species'].unique()
```

Out[13]:
```
array(['Iris-setosa', 'Iris-versicolor'], dtype=object)
```

In [14]:
```
# 1. by replace Pandas function
df['Species']=df['Species'].replace({'Iris-setosa':0, 'Iris-versicolor':1, 'I
df
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_4696/237209451.py:2: SettingWith
CopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  df['Species']=df['Species'].replace({'Iris-setosa':0, 'Iris-versicolor':1,
'Iris-virginica':2})
```

Out[14]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
95	96	5.7	3.0	4.2	1.2	1
96	97	5.7	2.9	4.2	1.3	1
97	98	6.2	2.9	4.3	1.3	1
98	99	5.1	2.5	3.0	1.1	1
99	100	5.7	2.8	4.1	1.3	1

100 rows × 6 columns

In [15]:
```
# 2.  Label Encoder
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Species'] = le.fit_transform(df['Species'])
df.tail(3)
```

```
C:\Users\omkan\AppData\Local\Temp/ipykernel_4696/3779767555.py:4: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/
stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pand
as.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-v
ersus-a-copy)
  df['Species'] = le.fit_transform(df['Species'])
```

Out[15]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
97	98	6.2	2.9	4.3	1.3	1
98	99	5.1	2.5	3.0	1.1	1
99	100	5.7	2.8	4.1	1.3	1

5. Define Variables x & y

x [indipendent variable, feature, input]

&

y [dependent variable, target, Label, output]

In [16]:
```
x=df.iloc[:,1:5]
y=df.iloc[:,5:]
x.head(),y.head()
```

Out[16]:
```
(   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
 0            5.1           3.5            1.4           0.2
 1            4.9           3.0            1.4           0.2
 2            4.7           3.2            1.3           0.2
 3            4.6           3.1            1.5           0.2
 4            5.0           3.6            1.4           0.2,
    Species
 0        0
 1        0
 2        0
 3        0
 4        0)
```

6.Feature Scaling -

Feature Scaling in x dependent variable

6.1. from sklearn.preprocessing import StandardScaler

ss=StandardScaler() ssx=ss.fit_transform(x) print(ssx) print(pd.DataFrame(ssx).describe())

6.2. from sklearn.preprocessing import MinMaxScaler

mms=MinMaxScaler() mmsx=mms.fit_transform(x) print(mmsx)
print(pd.DataFrame(mmsx).describe())

- https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/ (https://www.geeksforgeeks.org/standardscaler-minmaxscaler-and-robustscaler-techniques-ml/)
- https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/ (https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/)
- https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler (https://stackoverflow.com/questions/40758562/can-anyone-explain-me-standardscaler)
- https://machinelearninggeek.com/feature-scaling-minmax-standard-and-robust-scaler/ (https://machinelearninggeek.com/feature-scaling-minmax-standard-and-robust-scaler/)

"'StandardScaler follows Standard Normal Distribution (SND). Therefore, it makes mean = 0 and scales the data to unit variance. MinMaxScaler scales all the data features in the range [0, 1] or else in the range [-1, 1] if there are negative values in the dataset.'"

In [17]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
ssx=ss.fit_transform(x)
print(ssx[:5])
print(pd.DataFrame(ssx).describe())
```

```
[[-0.5810659   0.85713543 -1.01435952 -1.03824799]
 [-0.89430898 -0.19845007 -1.01435952 -1.03824799]
 [-1.20755205  0.22378413 -1.08374115 -1.03824799]
 [-1.36417359  0.01266703 -0.94497788 -1.03824799]
 [-0.73768744  1.06825253 -1.01435952 -1.03824799]]
                   0             1             2             3
count  1.000000e+02  1.000000e+02  1.000000e+02  1.000000e+02
mean  -1.547096e-15  2.176037e-16 -3.019807e-16  4.818368e-16
std    1.005038e+00  1.005038e+00  1.005038e+00  1.005038e+00
min   -1.834038e+00 -2.309621e+00 -1.291886e+00 -1.215726e+00
25%   -7.376874e-01 -6.206843e-01 -9.449779e-01 -1.038248e+00
50%   -1.112013e-01 -9.289152e-02 -2.858523e-01  2.662174e-02
75%    6.719064e-01  6.460183e-01  1.015053e+00  9.140132e-01
max    2.394743e+00  2.757189e+00  1.552761e+00  1.801405e+00
```

In [18]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
mmsx=mms.fit_transform(x)
print(mmsx[:5])
print(pd.DataFrame(mmsx).describe())
```

```
[[0.2962963  0.625      0.09756098 0.05882353]
 [0.22222222 0.41666667 0.09756098 0.05882353]
 [0.14814815 0.5        0.07317073 0.05882353]
 [0.11111111 0.45833333 0.12195122 0.05882353]
 [0.25925926 0.66666667 0.09756098 0.05882353]]
                0           1           2           3
count  100.000000  100.000000  100.000000  100.000000
mean     0.433704    0.455833    0.454146    0.402941
std      0.237666    0.198357    0.353308    0.333110
min      0.000000    0.000000    0.000000    0.000000
25%      0.259259    0.333333    0.121951    0.058824
50%      0.407407    0.437500    0.353659    0.411765
75%      0.592593    0.583333    0.810976    0.705882
max      1.000000    1.000000    1.000000    1.000000
```

7. LabelBinarizer -

convert y Label into binary

7.1. from sklearn.preprocessing import LabelBinarizer

- lb = LabelBinarizer()
- lby=lb.fit_transform(y)
- print(lby)
- https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html (https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelBinarizer.html)
-

7.2. from keras.utils import np_utils

- npy=np_u
- npy=utils.to_categorical(y)
- print(npy)
- https://stackoverflow.com/questions/60218142/error-when-doing-import-tensorflow-keras-utils-np-utils (https://stackoverflow.com/questions/60218142/error-when-doing-import-tensorflow-keras-utils-np-utils)
-

7.3. import pandas as pd

- y=pd.Series(y)
- pdy=pd.get_dummies(y).values
- print(pdy.head())

or

- ys=y.squeeze()
- pdy=pd.get_dummies(ys).values
- print(pdy[:5])
- https://www.geeksforgeeks.org/python-pandas-get_dummies-method/ (https://www.geeksforgeeks.org/python-pandas-get_dummies-method/)
-

-
- https://www.geeksforgeeks.org/how-to-convert-categorical-data-to-binary-data-in-python/ (https://www.geeksforgeeks.org/how-to-convert-categorical-data-to-binary-data-in-python/)

In [19]:
```
# 1.
from sklearn.preprocessing import LabelBinarizer
lb = LabelBinarizer()
lby=lb.fit_transform(y)
print(lby[:5])
```

```
[[0]
 [0]
 [0]
 [0]
 [0]]
```

In [20]:
```
#2.
from keras.utils import np_utils
npy=np_utils.to_categorical(y)
print(npy[:5])
```

```
[[1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]
 [1. 0.]]
```

In [ ]:
```

```

convert dataframe to series

- https://datatofish.com/pandas-dataframe-to-series/ (https://datatofish.com/pandas-dataframe-to-series/)
- You can convert Pandas DataFrame to a Series using squeeze:
- df.squeeze()

In [21]:
```
y[:2]
```

Out[21]:

	Species
0	0
1	0

In [22]:
```
# 3.
import pandas as pd
ys=y.squeeze()
pdy=pd.get_dummies(ys).values
print(pdy[:5])
```

```
[[1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]]
```

In [23]:
```
#3.1
import pandas as pd
pdydf=pd.get_dummies(ys)
print(pdydf.head(2))
```

```
   0  1
0  1  0
1  1  0
```

8. train_test_split (Split data set in train and test)

In [24]:
```
from sklearn.model_selection import train_test_split
```

In [25]:
```
x_train,x_test,y_train,y_test=train_test_split(ssx,pdy,test_size=.25, random_
```

In [26]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(75, 4)
(25, 4)
(75, 2)
(25, 2)
```

In [ ]:
```

```

9. Import Liberaries & Build NEURAL NETWORK Model

----------REQUIRED TO READ----------
https://www.javatpoint.com/artificial-neural-network (https://www.javatpoint.com/artificial-neural-network)

https://blog.knoldus.com/getting-familiar-with-activation-function-and-its-types/ (https://blog.knoldus.com/getting-familiar-with-activation-function-and-its-types/)

----------EXTRA----------

- https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/ (https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/)
- https://dotnettutorials.net/lesson/architecture-of-artificial-neural-network/ (https://dotnettutorials.net/lesson/architecture-of-artificial-neural-network/)
- https://medium.com/mlearning-ai/several-types-of-artificial-neural-networks-architecture-that-you-should-know-c169a5e22ec7 (https://medium.com/mlearning-ai/several-types-of-artificial-neural-networks-architecture-that-you-should-know-c169a5e22ec7)
- https://www.tutorialspoint.com/tensorflow/tensorflow_keras.htm#:~:text=Keras%20is%20cor level%20Python,can%20be%20of%20the%20following%20two%20types%20%E2%88%92 (https://www.tutorialspoint.com/tensorflow/tensorflow_keras.htm#:~:text=Keras%20is%20co level%20Python,can%20be%20of%20the%20following%20two%20types%20%E2%88%92)

9.1. import liberaries

In [27]:
```
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras.optimizers import Adam
from keras.metrics import binary_crossentropy
```

- https://www.tensorflow.org/api_docs/python/tf/keras (https://www.tensorflow.org/api_docs/python/tf/keras)
- https://keras.io/ (https://keras.io/)
- https://www.tutorialspoint.com/tensorflow/tensorflow_keras.htm (https://www.tutorialspoint.com/tensorflow/tensorflow_keras.htm)
-

Types of Artificial Neural Networks Architecture

9.2.1 Single layer feedforward neural networks

9.2.2 Multi layer feedforward neural networks

9.2.3 Multi layer Preceptron

9.2.4 Recurent Neural Networks / Feedback Neural Network

- https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/ (https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/)
- https://www.v7labs.com/blog/neural-network-architectures-guide (https://www.v7labs.com/blog/neural-network-architectures-guide)
- https://www.upgrad.com/blog/types-of-neural-networks/ (https://www.upgrad.com/blog/types-of-neural-networks/)
- https://www.geeksforgeeks.org/ml-architecture-and-learning-process-in-neural-network/ (https://www.geeksforgeeks.org/ml-architecture-and-learning-process-in-neural-network/)
- https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks (https://en.wikipedia.org/wiki/Types_of_artificial_neural_networks)
- https://medium.com/mlearning-ai/several-types-of-artificial-neural-networks-architecture-that-you-should-know-c169a5e22ec7 (https://medium.com/mlearning-ai/several-types-of-artificial-neural-networks-architecture-that-you-should-know-c169a5e22ec7)
- https://www.upgrad.com/blog/classification-model-using-artificial-neural-networks/#:~:text=Home%20%3E%20Artificial%20Intelligence%20%3E%20Classification%2 (https://www.upgrad.com/blog/classification-model-using-artificial-neural-networks/#:~:text=Home%20%3E%20Artificial%20Intelligence%20%3E%20Classification%2
- https://www.tensorflow.org/tutorials/structured_data/preprocessing_layers (https://www.tensorflow.org/tutorials/structured_data/preprocessing_layers)

In [28]:
```
model=Sequential([
    # input layer
    Dense(units=5,input_shape=(4,),activation='relu'),

    # hidden layer
  #  Dense(units=25,activation='sigmoid'),
  #  Dense(units=35,activation='sigmoid'),
  #  Dense(units=55,activation='sigmoid'),

    # output layers
    Dense(units=2,activation='sigmoid')  #  act.- sigmoid in binary classific
    # & output(units/output_dim) precepton - 2 two

])
```

Output Layers - Remember Note

- Dense(units=2,activation='sigmoid') act.- sigmoid in binary classification & output(units/output_dim) precepton - 2 two
- Dense(units= 3 or more, activation='softax') act.- softmax in multiclass classification & output(units/output_dim) precepton - tree or more put similar as much classes in y

9.3. Model Compile

- model.compile
- (loss='categorical_crossentropy {multiclass classification}'} (loss='binary_crossentropy {binary classificatio}'}
- optimizer='adam'
- metrics='accuracy')
-
-

> Multiclass --- model.compile(loss='categorical_crossentropy',optimizer='adam', metrics='accuracy')

> Binaryclass --- model.compile(optimizer='Adam',loss='binary_crossentropy', metrics=['accuracy'])

- https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/ (https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/)
- https://www.geeksforgeeks.org/adam-optimizer-in-tensorflow/ (https://www.geeksforgeeks.org/adam-optimizer-in-tensorflow/)
- https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/ (https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/)

- http://optimization.cbe.cornell.edu/index.php?title=Adam (http://optimization.cbe.cornell.edu/index.php?title=Adam)
-
- - https://github.com/tensorflow/tensorflow/issues/34003 (https://github.com/tensorflow/tensorflow/issues/34003)

In [29]:
```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics='accuracy')
```

9.4. Model Summery

In [30]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 5)                 25

 dense_1 (Dense)             (None, 2)                 12

=================================================================
Total params: 37
Trainable params: 37
Non-trainable params: 0
_________________________________________________________________
```

10. Train the Model

- 10.1 d=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100)
- 10.2 model.fit(x_train,y_train, batch_size=100,epochs=100)

In [31]:
```
d=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100)
```

```
racy: 0.9867 - val_loss: 0.3785 - val_accuracy: 1.0000
Epoch 95/100
3/3 [==============================] - 0s 17ms/step - loss: 0.3857 - accu
racy: 0.9867 - val_loss: 0.3748 - val_accuracy: 1.0000
Epoch 96/100
3/3 [==============================] - 0s 15ms/step - loss: 0.3820 - accu
racy: 0.9867 - val_loss: 0.3710 - val_accuracy: 1.0000
Epoch 97/100
3/3 [==============================] - 0s 21ms/step - loss: 0.3781 - accu
racy: 1.0000 - val_loss: 0.3673 - val_accuracy: 1.0000
Epoch 98/100
3/3 [==============================] - 0s 17ms/step - loss: 0.3744 - accu
racy: 1.0000 - val_loss: 0.3636 - val_accuracy: 1.0000
Epoch 99/100
3/3 [==============================] - 0s 16ms/step - loss: 0.3707 - accu
racy: 1.0000 - val_loss: 0.3600 - val_accuracy: 1.0000
Epoch 100/100
3/3 [==============================] - 0s 18ms/step - loss: 0.3670 - accu
racy: 1.0000 - val_loss: 0.3564 - val_accuracy: 1.0000
```

11. Model Prediction

- y_pred1=model.predict(x_test)
- y_pred1

In [32]:
```
y_pred1=model.predict(x_test)
y_pred1
```

```
1/1 [==============================] - 0s 71ms/step
```

Out[32]:
```
array([[0.35632122, 0.79355645],
       [0.45324525, 0.21426894],
       [0.25447538, 0.84117913],
       [0.42536646, 0.786521  ],
       [0.413159  , 0.11853209],
       [0.31681812, 0.8682604 ],
       [0.2372292 , 0.86821324],
       [0.420932  , 0.13377081],
       [0.4658677 , 0.13000166],
       [0.28528187, 0.84965974],
       [0.37201688, 0.80710095],
       [0.51269567, 0.12046862],
       [0.42344606, 0.1390244 ],
       [0.22992049, 0.9079846 ],
       [0.5103609 , 0.14275613],
       [0.5287743 , 0.21951197],
       [0.2993209 , 0.88676655],
       [0.5151383 , 0.16925743],
       [0.26059264, 0.8918561 ],
       [0.31687665, 0.8731712 ],
       [0.29762527, 0.8641351 ],
       [0.50577   , 0.1379226 ],
       [0.47631812, 0.22830497],
       [0.27522358, 0.88604224],
       [0.45224497, 0.11221167]], dtype=float32)
```

Model prediction coparision with y_test & Predict

In [33]:
```
y_test_class=np.argmax(y_test,axis=1)
y_pred1_class=np.argmax(y_pred1,axis=1)
y_pred1_class,y_test_class
```

Out[33]:
```
(array([1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
        0, 1, 0], dtype=int64),
 array([1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0,
        0, 1, 0], dtype=int64))
```

12 Model Evaluation

- from sklearn.metrics import classification_report,confusion_matrix,accuracy_score

In [34]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_s
```

In [35]:
```
print(confusion_matrix(y_test_class,y_pred1_class))
print(classification_report(y_test_class,y_pred1_class))
print(accuracy_score(y_test_class,y_pred1_class))
```

```
[[12  0]
 [ 0 13]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      1.00      1.00        13

    accuracy                           1.00        25
   macro avg       1.00      1.00      1.00        25
weighted avg       1.00      1.00      1.00        25

1.0
```

a. Artificial Neural Network - CLASSIFICATION

Classification Models using Artificial Neural Networks (ANN) -- (Artificial Neural Network Algorithm - CLASSIFICATION Models)

1. ANN Binary Classification

2. ANN multiclass Classification

- articals
- - https://www.analyticsvidhya.com/blog/2021/07/demystifying-the-difference-between-multi-class-and-multi-label-classification-problem-statements-in-deep-learning/ (https://www.analyticsvidhya.com/blog/2021/07/demystifying-the-difference-between-multi-class-and-multi-label-classification-problem-statements-in-deep-learning/)
- - https://machinelearningmastery.com/multi-label-classification-with-deep-learning/ (https://machinelearningmastery.com/multi-label-classification-with-deep-learning/)

2. ANN Multiclass Classification

- here should be three OR more class in categorical column that will used in y (output, targrt, label & depentent variable)
  - three OR more class in y features

In [36]:
```
df=iris
df.shape,df.head()
```

Out[36]:
```
((150, 6),
    Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm      Species
 0   1            5.1           3.5            1.4           0.2  Iris-setosa
 1   2            4.9           3.0            1.4           0.2  Iris-setosa
 2   3            4.7           3.2            1.3           0.2  Iris-setosa
 3   4            4.6           3.1            1.5           0.2  Iris-setosa
 4   5            5.0           3.6            1.4           0.2  Iris-setosa)
```

In [37]:
```
print(df['Species'].unique())
```

```
['Iris-setosa' 'Iris-versicolor' 'Iris-virginica']
```

In [38]:
```
### 4. LABEL Encodinging
# 4.1. by replace Pandas function
df['Species']=df['Species'].replace({'Iris-setosa':0, 'Iris-versicolor':1, 'I
df[:3]
```

Out[38]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0

In [39]:
```
print(df['Species'].unique())
```

```
[0 1 2]
```

In [40]:
```
### 5. Define Variables x & y
#### x [  indipendent variable, feature, input  ]
####           &
#### y [  dependent variable, target, Label, output  ]

x=df.iloc[:,1:5].values
y=df.iloc[:,5:].values
x[:2],y[148:]
```

Out[40]:
```
(array([[5.1, 3.5, 1.4, 0.2],
        [4.9, 3. , 1.4, 0.2]]),
 array([[2],
        [2]], dtype=int64))
```

In [41]:
```
### 6.Feature Scaling -
## Feature Scaling in x dependent variable
#### 6.1.  from sklearn.preprocessing import StandardScaler
#### 6.2. from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
mmsx=mms.fit_transform(x)
print(mmsx[:5])
print(pd.DataFrame(mmsx).describe())
```

```
[[0.22222222 0.625      0.06779661 0.04166667]
 [0.16666667 0.41666667 0.06779661 0.04166667]
 [0.11111111 0.5        0.05084746 0.04166667]
 [0.08333333 0.45833333 0.08474576 0.04166667]
 [0.19444444 0.66666667 0.06779661 0.04166667]]
                0           1           2           3
count  150.000000  150.000000  150.000000  150.000000
mean     0.428704    0.439167    0.467571    0.457778
std      0.230018    0.180664    0.299054    0.317984
min      0.000000    0.000000    0.000000    0.000000
25%      0.222222    0.333333    0.101695    0.083333
50%      0.416667    0.416667    0.567797    0.500000
75%      0.583333    0.541667    0.694915    0.708333
max      1.000000    1.000000    1.000000    1.000000
```

In [42]:
```
### 7. LabelBinarizer -
## convert  y  Label  into binary
#### 7.1. from sklearn.preprocessing import LabelBinarizer
#### 7.2. from keras.utils import np_utils
#### 7.3. import pandas as pd

#2.
from keras.utils import np_utils
npy=np_utils.to_categorical(y)
print(npy[:5])
```

```
[[1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

In [43]:
```
## 8. train_test_split  (Split data set in train and test)
from sklearn.model_selection import train_test_split
#x_train,x_test,y_train,y_test=train_test_split(mmsx,y,test_size=.25, random_s
x_train,x_test,y_train,y_test=train_test_split(mmsx,npy,test_size=.25, random_
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(112, 4)
(38, 4)
(112, 3)
(38, 3)
```

In [44]:
```
## 9. Import Liberaries & Build NEURAL NETWORK  Model
### 9.1. import liberaries
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense,Activation
from keras.optimizers import Adam
from keras.metrics import categorical_crossentropy
```

In [45]:
```
## Types of Artificial Neural Networks Architecture
### 9.2.1 Single layer feedforward neural networks
### 9.2.2 Multi layer feedforward neural networks
### 9.2.3 Multi layer Preceptron
### 9.2.4 Recurent Neural Networks / Feedback Neural Network

# building tje model
model= Sequential() # here we build sequential model
# input layer/first layer  - has 4 neuron or nodes , it could be 100 or 200
model.add(Dense(4,input_shape=(4,),activation='relu'))
'''
# model.add(Dense((4-nerons/nodes),input_shape=(4,-input feature amt.
alweys use coma , represent of input as one dientional array - ),
actiation='relu'- relu is rectifire linear unit function-that means if
it get any negative value, it will ake it zero else pass the alue at it is))
'''
# hidden layer

model.add(Dense(14,input_shape=(4,),activation='relu'))
model.add(Dense(14,input_shape=(4,),activation='relu'))

# output Layer
model.add(Dense(3,activation='softmax')) # if label/classes more then two
''' here use three nodes because it has three columns in binary, here using a
funtion - softmax because  here label/classes are three
'''
 #  act.- softmax in multiclass classification
    # & output(units/output_dim) precepton - tree or more put similar as much
1
```

Out[45]: 1

In [46]:
```
### 9.3. Model Compile
# Multiclass
#model.compile(loss='categorical_crossentropy',optimizer='adam', metrics='acc
# compile the model
model.compile(optimizer='Adam',loss='categorical_crossentropy', metrics=['acc
''' we useing here adam optimizer. other stochatice gredient desent
   sparse_categorical_crossentropy
'''
1
```

Out[46]: 1

In [47]:
```
### 9.4. Model Summery
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 4)                 20

 dense_3 (Dense)             (None, 14)                70

 dense_4 (Dense)             (None, 14)                210

 dense_5 (Dense)             (None, 3)                 45

=================================================================
Total params: 345
Trainable params: 345
Non-trainable params: 0
_________________________________________________________________
```

In [48]:
```
## 10. Train the Model
#- 10.1 d=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10(
#- 10.2 model.fit(x_train,y_train, batch_size=100,epochs=100) batch_size -  DA
model.fit(x_train,y_train, batch_size=100,epochs=100)
```

```
Epoch 62/100
2/2 [==============================] - 0s 4ms/step - loss: 0.8412 - accur
acy: 0.6786
Epoch 63/100
2/2 [==============================] - 0s 3ms/step - loss: 0.8326 - accur
acy: 0.6786
Epoch 64/100
2/2 [==============================] - 0s 3ms/step - loss: 0.8225 - accur
acy: 0.6786
Epoch 65/100
2/2 [==============================] - 0s 3ms/step - loss: 0.8119 - accur
acy: 0.6786
Epoch 66/100
2/2 [==============================] - 0s 3ms/step - loss: 0.7998 - accur
acy: 0.6786
Epoch 67/100
2/2 [==============================] - 0s 3ms/step - loss: 0.7863 - accur
acy: 0.6786
Epoch 68/100
2/2 [==============================] - 0s 3ms/step - loss: 0.7724 - accur
```

In [49]:
```
## 11. Model Prediction
y_pred1=model.predict(x_test)
y_pred1[:5]
```

```
2/2 [==============================] - 0s 1ms/step
```

Out[49]:
```
array([[0.9285499 , 0.05483145, 0.01661871],
       [0.08018914, 0.4111649 , 0.508646  ],
       [0.07539763, 0.41908064, 0.5055217 ],
       [0.89437073, 0.07866919, 0.02696015],
       [0.9168644 , 0.06218121, 0.02095439]], dtype=float32)
```

In [50]:
```
# Model prediction comparision with y_test & Predict
# np.argmax(y_test,axis=1) - we use this because y in binary
y_test_class=np.argmax(y_test,axis=1)
y_pred1_class=np.argmax(y_pred1,axis=1)
y_pred1_class,y_test_class
```

Out[50]:
```
(array([0, 2, 2, 0, 0, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 0, 2, 2, 0, 2, 2, 0,
        2, 1, 0, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 0, 2, 0], dtype=int64),
 array([0, 1, 1, 0, 0, 1, 2, 1, 1, 2, 2, 1, 1, 1, 2, 0, 1, 2, 0, 2, 1, 0,
        1, 1, 0, 0, 2, 2, 2, 1, 0, 2, 1, 2, 0, 0, 2, 0], dtype=int64))
```

In [51]:
```
## 12 Model Evaluation
from sklearn.metrics import classification_report,confusion_matrix,accuracy_s

print(confusion_matrix(y_test_class,y_pred1_class))
print(classification_report(y_test_class,y_pred1_class))
print(accuracy_score(y_test_class,y_pred1_class))
```

```
[[12  0  0]
 [ 0  2 12]
 [ 0  0 12]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        12
           1       1.00      0.14      0.25        14
           2       0.50      1.00      0.67        12

    accuracy                           0.68        38
   macro avg       0.83      0.71      0.64        38
weighted avg       0.84      0.68      0.62        38

0.6842105263157895
```

In [52]:
```
(12+0+12)/38    ## accuracy
```

Out[52]:
```
0.631578947368421
```

```
## 12 Model Evaluation
from sklearn.metrics import
classification_report,confusion_matrix,accuracy_score
```

```
print(confusion_matrix(y_test_class,y_pred1))
print(classification_report(y_test_class,y_pred1))
print(accuracy_score(y_test_class,y_pred1))
```

In [ ]:

In [ ]:

b. Artificial Neural Network - REGRESSION

- b.1. import keras libery and funtions
- b.2. from sklearn.neural_network import MLPRegressor
- -
- -
- - https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html)
- - https://stackoverflow.com/questions/55786860/how-is-the-hidden-layer-size-determined-for-mlpregressor-in-scikitlearn (https://stackoverflow.com/questions/55786860/how-is-the-hidden-layer-size-determined-for-mlpregressor-in-scikitlearn)
- - https://www.programcreek.com/python/example/93778/sklearn.neural_network.MLPRe (https://www.programcreek.com/python/example/93778/sklearn.neural_network.MLPRe

In [53]:
```
df
```

Out[53]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

150 rows × 6 columns

In [54]:
```
# Define x & y
x=df.iloc[:,1:3].values
y=df.iloc[:,4].values
x[:5],y[:5]
```

Out[54]:
```
(array([[5.1, 3.5],
        [4.9, 3. ],
        [4.7, 3.2],
        [4.6, 3.1],
        [5. , 3.6]]),
 array([0.2, 0.2, 0.2, 0.2, 0.2]))
```

In [55]:
```
# Split x & y in train_test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.25, random_stat
```

In [56]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x_train=mms.fit_transform(x_train)
x_test=mms.fit_transform(x_test)
x_train[:5],x_test[:5]
```

Out[56]:
```
(array([[0.22222222, 0.70833333],
        [0.63888889, 0.375     ],
        [0.86111111, 0.33333333],
        [0.        , 0.41666667],
        [0.13888889, 0.58333333]]),
 array([[0.        , 0.66666667],
        [0.41935484, 0.4       ],
        [0.29032258, 0.13333333],
        [0.06451613, 0.46666667],
        [0.25806452, 0.66666667]]))
```

In [57]:
```
print(pd.DataFrame(x_train).describe())
print(pd.DataFrame(x_test).describe())
```

```
                0           1
count  112.000000  112.000000
mean     0.429563    0.438616
std      0.235110    0.192528
min      0.000000    0.000000
25%      0.222222    0.333333
50%      0.416667    0.416667
75%      0.590278    0.541667
max      1.000000    1.000000
                0          1
count  38.000000  38.000000
mean    0.398132   0.438596
std     0.252365   0.227604
min     0.000000   0.000000
25%     0.193548   0.283333
50%     0.403226   0.400000
75%     0.548387   0.583333
max     1.000000   1.000000
```

In [58]:
```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(112, 2)
(38, 2)
(112,)
(38,)
```

In [59]:
```
# import keras libery and funtions
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras import metrics
```

In [60]:
```
# first input and first hidden layer
model = Sequential()
model.add(Dense(1,input_dim=2, activation='relu'))

# second hidden layer
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(10, activation='relu'))

# output layer
model.add(Dense(1, activation='linear'))
'''
regression so use linear activation and one preceptron
because output in single value.
'''
```

Out[60]:
```
'\nregression so use linear activation and one preceptron\nbecause output in
single value.\n'
```

In [61]:
```
# compile ANN
model.compile(optimizer= 'Adam', loss='mean_squared_error',metrics=['accuracy
```

In [62]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 1)                 3

 dense_7 (Dense)             (None, 10)                20

 dense_8 (Dense)             (None, 10)                110

 dense_9 (Dense)             (None, 10)                110

 dense_10 (Dense)            (None, 1)                 11

=================================================================
Total params: 254
Trainable params: 254
Non-trainable params: 0
_________________________________________________________________
```

In [63]:
```
##  TRAIN THE MODEL
# fit and display the summary
#model.fit(x_train,y_train, batch_size=100,epochs=100)
model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=100)
#model.fit(x_train,y_train,epochs=1000,verbose=1)
#verbose=1 paraeter for visulise epoch
```

```
Epoch 95/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1318 - accu
racy: 0.0446 - val_loss: 0.1810 - val_accuracy: 0.0526
Epoch 96/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1312 - accu
racy: 0.0446 - val_loss: 0.1808 - val_accuracy: 0.0526
Epoch 97/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1310 - accu
racy: 0.0446 - val_loss: 0.1800 - val_accuracy: 0.0526
Epoch 98/100
4/4 [==============================] - 0s 12ms/step - loss: 0.1307 - accu
racy: 0.0446 - val_loss: 0.1791 - val_accuracy: 0.0526
Epoch 99/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1307 - accu
racy: 0.0446 - val_loss: 0.1778 - val_accuracy: 0.0526
Epoch 100/100
4/4 [==============================] - 0s 11ms/step - loss: 0.1300 - accu
racy: 0.0446 - val_loss: 0.1786 - val_accuracy: 0.0526
```

Out[63]:
```
<keras.callbacks.History at 0x2af0e5f5760>
```

In [64]:
```
# given 100 epochs in 4 layers
```

In [65]:
```
# Model prediction / testing the test dataset
y_pred_r=model.predict(x_test)
print(y_test[:5],y_pred_r[:5])
```

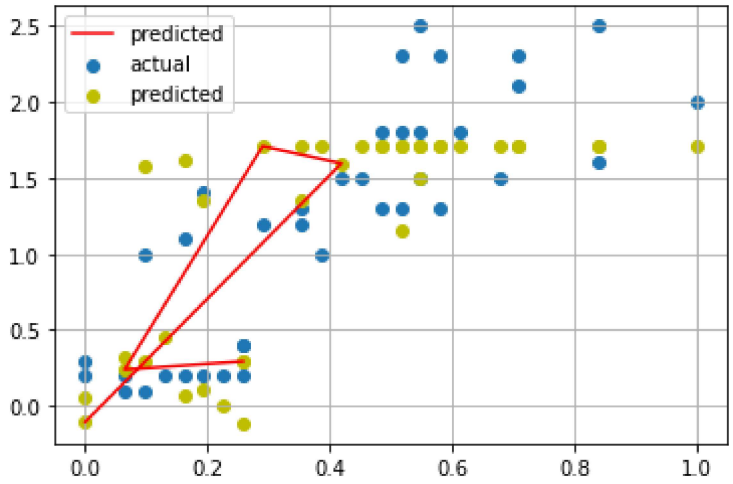
```
2/2 [==============================] - 0s 3ms/step
[0.3 1.5 1.2 0.2 0.4] [[-0.10434344]
 [ 1.5950947 ]
 [ 1.7084649 ]
 [ 0.24040714]
 [ 0.29407766]]
```

In [66]:
```
# model evaluation
from sklearn import metrics

mae=metrics.mean_absolute_error(y_test,y_pred_r)
mse=metrics.mean_squared_error(y_pred_r,y_test)
r2sq=metrics.r2_score(y_pred_r,y_test)

print(mae,mse,r2sq)
```

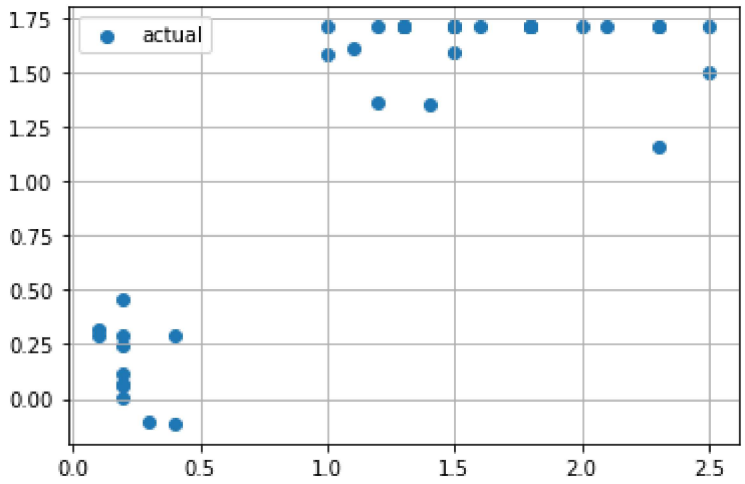
```
0.3295591668078774 0.17855632936833854 0.6396754621905472
```

visualization

In [67]:
```
import matplotlib.pyplot as plt
plt.scatter(x=x_test[:,0],y=y_test,label ='actual')
plt.scatter(x_test[:,0],y_pred_r, c='y',label ='predicted')
plt.plot(x_test[:5,0],y_pred_r[:5], c='r',label ='predicted')
plt.legend()
plt.grid()
plt.show()
```



In [68]:
```
import matplotlib.pyplot as plt
plt.scatter(x=y_test,y=y_pred_r,label ='actual')
plt.legend()
plt.grid()
plt.show()
```



In [ ]:

In [69]:
```
from sklearn import metrics
```

In [70]:
```
mae=metrics.mean_absolute_error(y_test,y_pred_r)
mse=metrics.mean_squared_error(y_pred_r,y_test)
r2sq=metrics.r2_score(y_pred_r,y_test)
```

In [71]:
```
print(mae,mse,r2sq)
```

```
0.3295591668078774 0.17855632936833854 0.6396754621905472
```

In [ ]:
```

```

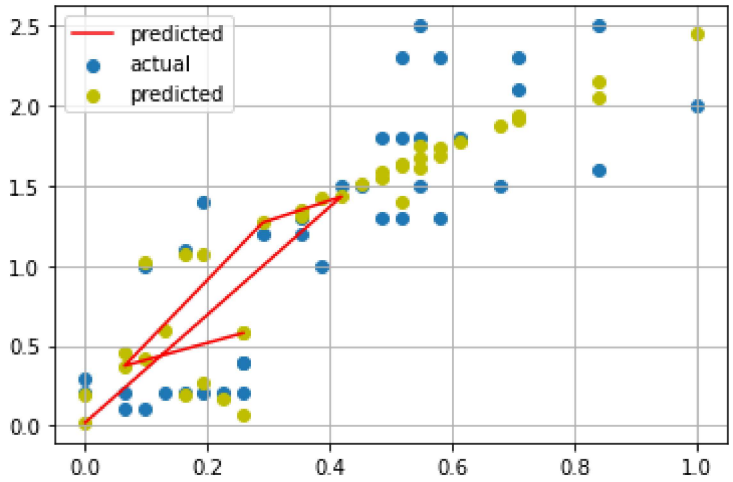
2. from sklearn.neural_network import MLPRegressor

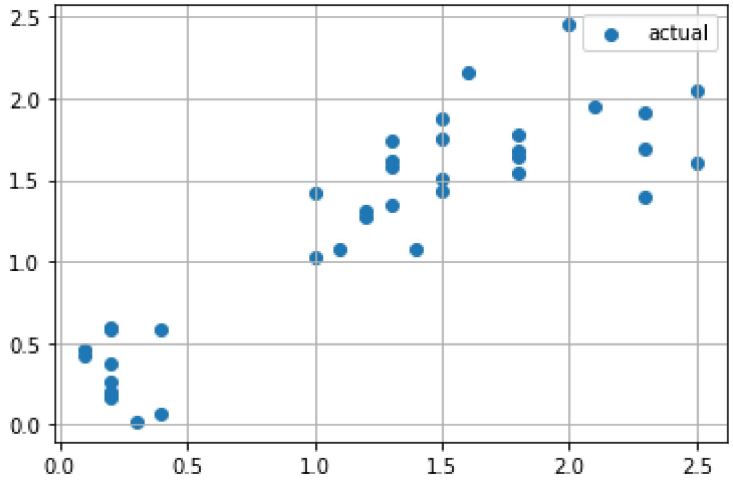
- https://stackoverflow.com/questions/55786860/how-is-the-hidden-layer-size-determined-for-mlpregressor-in-scikitlearn (https://stackoverflow.com/questions/55786860/how-is-the-hidden-layer-size-determined-for-mlpregressor-in-scikitlearn)

In [72]:
```
from sklearn.neural_network import MLPRegressor

mlpr=MLPRegressor(hidden_layer_sizes=(10,10,),activation='logistic',
                  max_iter=10, solver='lbfgs')

mlpr=MLPRegressor(hidden_layer_sizes=(10,10,),activation='relu',
                  max_iter=10, solver='lbfgs')

mlpr=MLPRegressor(hidden_layer_sizes=(10,10,),activation='relu',
                  max_iter=10, solver='lbfgs')

mlpr.fit(x_train, y_train)

y_pred_r=mlpr.predict(x_test)
print('y_test',y_test)
print('y_pred_r',y_pred_r)


from sklearn import metrics

mae=metrics.mean_absolute_error(y_test,y_pred_r)
mse=metrics.mean_squared_error(y_pred_r,y_test)
r2sq=metrics.r2_score(y_pred_r,y_test)

print(mae,mse,r2sq)

 #visualization
import matplotlib.pyplot as plt
plt.scatter(x=x_test[:,0],y=y_test,label ='actual')
plt.scatter(x_test[:,0],y_pred_r, c='y',label ='predicted')
plt.plot(x_test[:5,0],y_pred_r[:5], c='r',label ='predicted')
plt.legend()
plt.grid()
plt.show()
import matplotlib.pyplot as plt
plt.scatter(x=y_test,y=y_pred_r,label ='actual')
plt.legend()
plt.grid()
plt.show()
```

```
y_test [0.3 1.5 1.2 0.2 0.4 1.3 2.5 1.1 1.4 1.8 1.8 1.3 1.3 1.  2.3 0.2 1.2
2.5
 0.1 1.6 1.3 0.2 1.5 1.  0.2 0.4 2.1 2.3 2.  1.5 0.2 2.3 1.5 1.8 0.2 0.2
 1.8 0.1]
y_pred_r [0.01879973 1.43188419 1.26836063 0.37377016 0.57835958 1.35012241
 2.04977831 1.07475514 1.07792443 1.68014967 1.63926878 1.62232039
 1.73797894 1.42490007 1.68713379 0.59901265 1.30793907 1.60920071
 0.41878286 2.15657005 1.5814395  0.19542975 1.87451411 1.02297382
 0.16591297 0.0681939  1.94417908 1.40109535 2.45057186 1.50666185
 0.19720513 1.91104461 1.7479432  1.77885983 0.57835958 0.26741224
 1.54754274 0.46208517]
0.2697821053991374 0.1223741007647035 0.7240736052011382
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neural_network\_multilaye
r_perceptron.py:500: ConvergenceWarning: lbfgs failed to converge (status=
1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```





from sklearn.linear_model import LinearRegression

https://www.cuemath.com/data/correlation-and-regression/ (https://www.cuemath.com/data/correlation-and-regression/)

Regression analysis is used to determine the relationship between two variables such that the value of the unknown variable can be estimated using the knowledge of the known variables.

```
x=x_test[:,0]
y=y_test
```

```
x,y
```

In [73]:
```
#x_train[start_row0:stop_row4,start_column0] -start:stop:step
x_train[:4,0],x_train[:4],x_test[:4,0],x_test[:4]
```

Out[73]:
```
(array([0.22222222, 0.63888889, 0.86111111, 0.        ]),
 array([[0.22222222, 0.70833333],
        [0.63888889, 0.375     ],
        [0.86111111, 0.33333333],
        [0.        , 0.41666667]]),
 array([0.        , 0.41935484, 0.29032258, 0.06451613]),
 array([[0.        , 0.66666667],
        [0.41935484, 0.4       ],
        [0.29032258, 0.13333333],
        [0.06451613, 0.46666667]]))
```

In [74]:
```
y_train[:4],y_train[:4],y_test[:4],y_test[:4]
```

Out[74]:
```
(array([0.4, 1.3, 1.9, 0.1]),
 array([0.4, 1.3, 1.9, 0.1]),
 array([0.3, 1.5, 1.2, 0.2]),
 array([0.3, 1.5, 1.2, 0.2]))
```

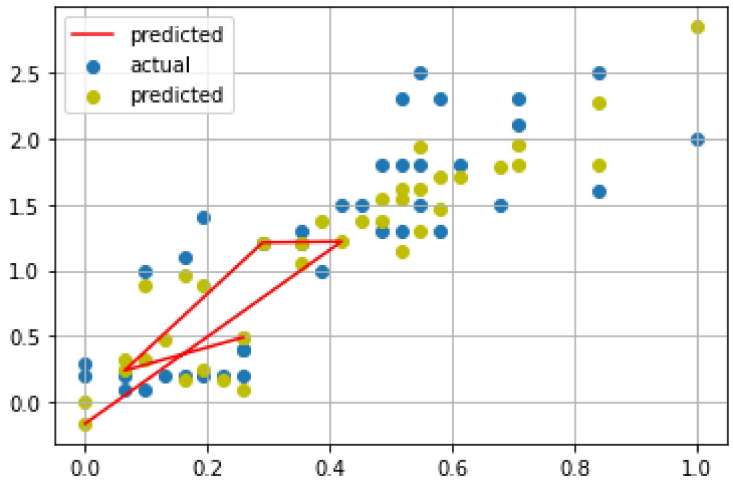
In [75]:
```
# mlr
from sklearn.linear_model import LinearRegression
model =LinearRegression()
model.fit(x_train,y_train)
print(model.score(x_train,y_train))
print('----------------------------------------')
y_pred=model.predict(x_test)
print(y_test[:4])
print(y_pred[:4])
print('----------------------------------------')
print(model.coef_[0],model.intercept_)
print(model.coef_[1],model.intercept_)
print(model.coef_,model.intercept_)
print('----------------------------------------')
print(x_test[:1],x_test[:1,0])
print('----------------------------------------')
print('y=b1*x1+b2*x2+c')
mlr_1=(2.533468405579969*0.08333333+-1.1960867374603343*0.58333333+0.63455161
print(mlr_1)
mlr_=(x_test[:1,0]*model.coef_[0]+x_test[:1,1]*model.coef_[1]+model.intercept_
print(mlr_)
```

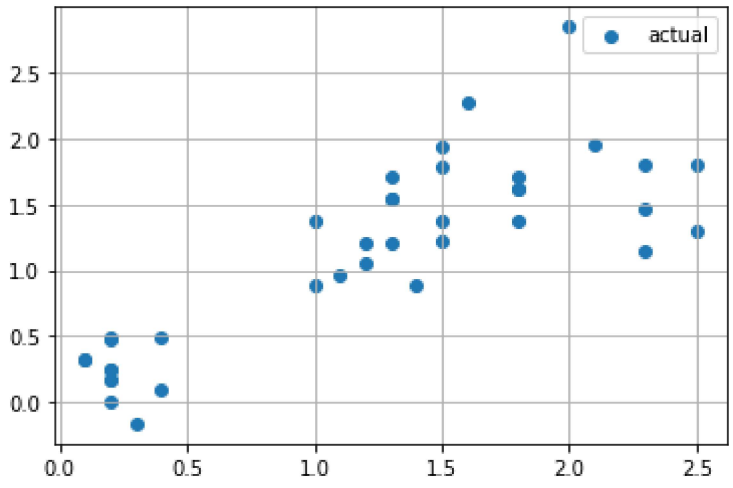
```
0.7542547834376911
----------------------------------------
[0.3 1.5 1.2 0.2]
[-0.16283954  1.21853916  1.21059647  0.23982738]
----------------------------------------
2.533468405579969 0.6345516162534967
-1.1960867374603343 0.6345516162534967
[ 2.53346841 -1.19608674] 0.6345516162534967
----------------------------------------
[[0.         0.66666667]] [0.]
----------------------------------------
y=b1*x1+b2*x2+c
0.14795671540869348
[-0.16283954]
```

In [76]:
```
from sklearn import metrics

mae=metrics.mean_absolute_error(y_test,y_pred)
mse=metrics.mean_squared_error(y_pred,y_test)
r2sq=metrics.r2_score(y_pred,y_test)
print(mae,mse,r2sq)
```

```
0.3308054213888762 0.19470351690821103 0.6096798363761641
```

In [77]:
```
import matplotlib.pyplot as plt
plt.scatter(x=x_test[:,0],y=y_test,label ='actual')
plt.scatter(x_test[:,0],y_pred, c='y',label ='predicted')
plt.plot(x_test[:5,0],y_pred[:5], c='r',label ='predicted')
plt.legend()
plt.grid()
plt.show()

import matplotlib.pyplot as plt
plt.scatter(x=y_test,y=y_pred,label ='actual')
plt.legend()
plt.grid()
plt.show()
```





Om Kant Sharma