**EX -1** **Impute missing values in data inputs**

**Aim:**

To Implement missing values in data input

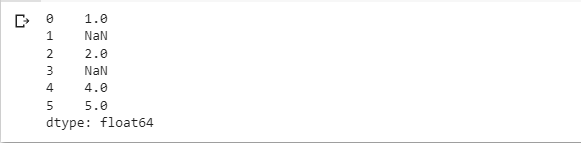
**Program:**

import pandas as pd

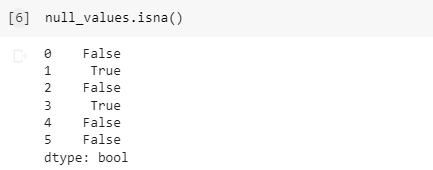
import numpy as np

null\_values=pd.Series([1,np.NaN,2,np.NaN,4,5])

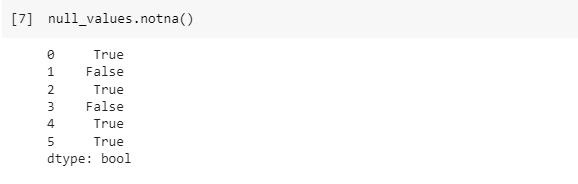
null\_values



null\_values.isna()



null\_values.notna()



null\_values.fillna(0)



null\_values.fillna(np.mean(null\_values))



null\_values.interpolate()



**Result:**

**EX -2** Use feature selection/extraction method to perform dimensionality reduction

**Aim:**

To use feature selection/extraction method to perform dimensionality reduction

**Program:**

from google.colab import files

uploaded =files.upload()



from pandas import read\_csv

from numpy import set\_printoptions

from sklearn.feature\_selection import SelectKBest

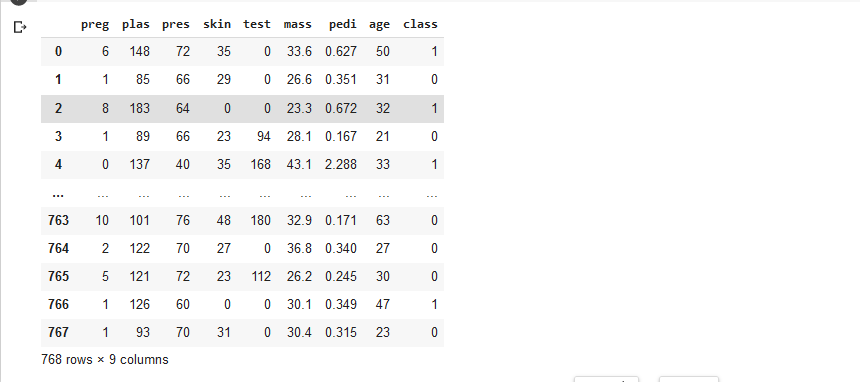
from sklearn.feature\_selection import f\_classif

filename = '/content/pima-indians-diabetes.csv'

# names =['preg','plas','pres','skin','test','mass','pedi','age','class']

dataframe = read\_csv(filename,names=names)

dataframe



array = dataframe.values

X=array[:,0:8]

Y=array[:,8]

test = SelectKBest(score\_func=f\_classif,k=4)

fit = test.fit(X,Y)

set\_printoptions(precision=3)

print(fit.scores\_)

features = fit.transfrm(X)

# summarize selected features

print(features[0:5,:])



**Result:**

**EX -3** Demonstrate Naïve Bayes Classification

**Aim:**

To demonstrate Naïve Bayes Classification.

**Program:**

#load the iris ataset

from sklearn.datasets import load\_iris

iris = load\_iris()

#store the feature matrix (x) and response vector (y)

X = iris.data

Y = iris.target

#splitting x and y into training and testing sets

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test, Y\_train, Y\_test = train\_test\_split(X, Y, test\_size=0.4, random\_state=1)

#training the model on training set

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train, Y\_train)

#making predictions on the testing set

Y\_pred = gnb.predict(X\_test)

#comparing actual response values (Y\_test) with predicted response values (Y\_pred)

from sklearn import metrics

print("Gaussian Naive Bayes model accuracy (in %)",metrics.accuracy\_score(Y\_test, Y\_pred)\*100)

**Output:**

****

**Result:**

**EX-4** Classify the input dataset using decision tree

**Aim:**

To classify the input dataset using decision tree

**Program:**

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from google.colab import files

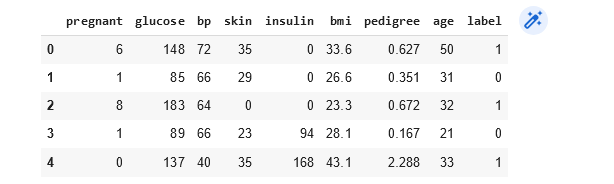
uploaded=files.upload()



col\_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']

pima = pd.read\_csv("diabetes.csv", header=0, names=col\_names)

pima.head()



feature\_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']

X = pima[feature\_cols]

y = pima.label

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

clf = DecisionTreeClassifier()

clf = clf.fit(X\_train,y\_train)

y\_pred = clf.predict(X\_test)

print("Accuracy in testing set:",metrics.accuracy\_score(y\_test, y\_pred))

****

from sklearn.tree import export\_graphviz

from six import StringIO

from IPython.display import Image

import pydotplus

dot\_data = StringIO()

export\_graphviz(clf, out\_file=dot\_data,

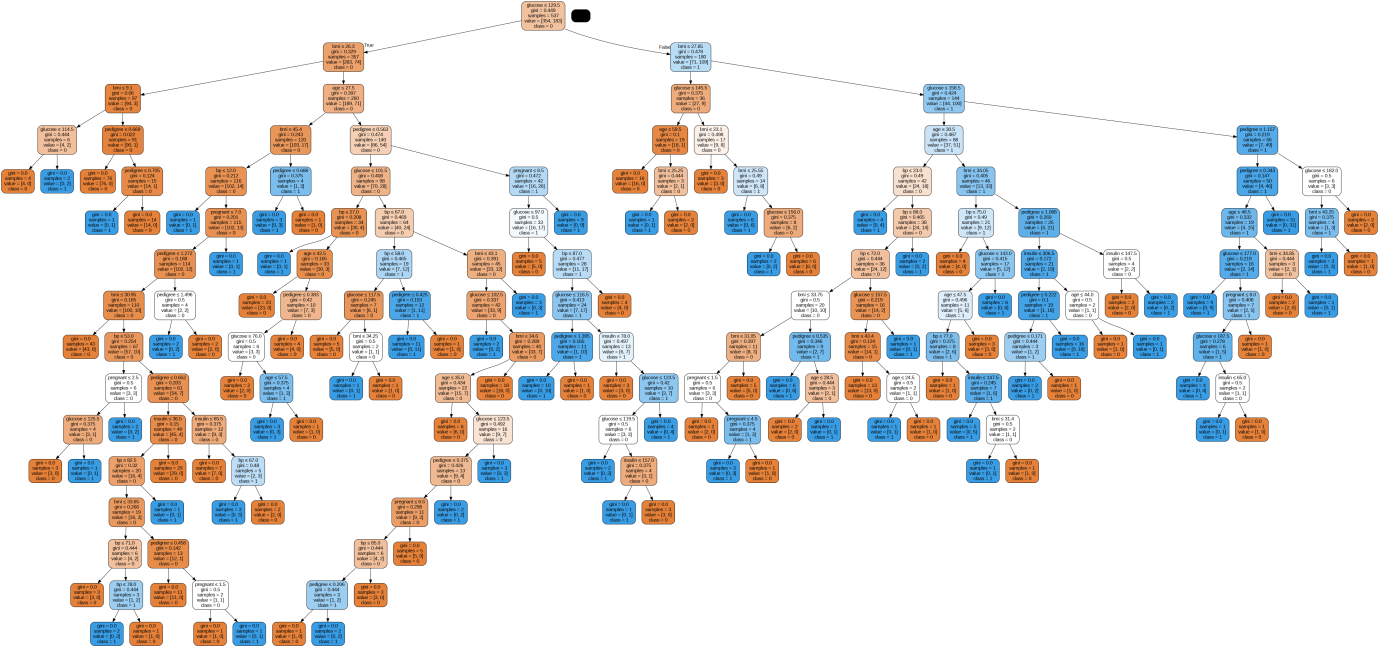
                    filled=True, rounded=True,

                    special\_characters=True, feature\_names = feature\_cols,class\_names=['0','1'])

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('diabetes.png')

Image(graph.create\_png())



clf = DecisionTreeClassifier(criterion="entropy", max\_depth=3)

clf = clf.fit(X\_train,y\_train)

y\_pred\_test = clf.predict(X\_test)

print("Accuracy:",metrics.accuracy\_score(y\_test, y\_pred\_test))



from six import StringIO

from IPython.display import Image

from sklearn.tree import export\_graphviz

import pydotplus

dot\_data = StringIO()

export\_graphviz(clf, out\_file=dot\_data,

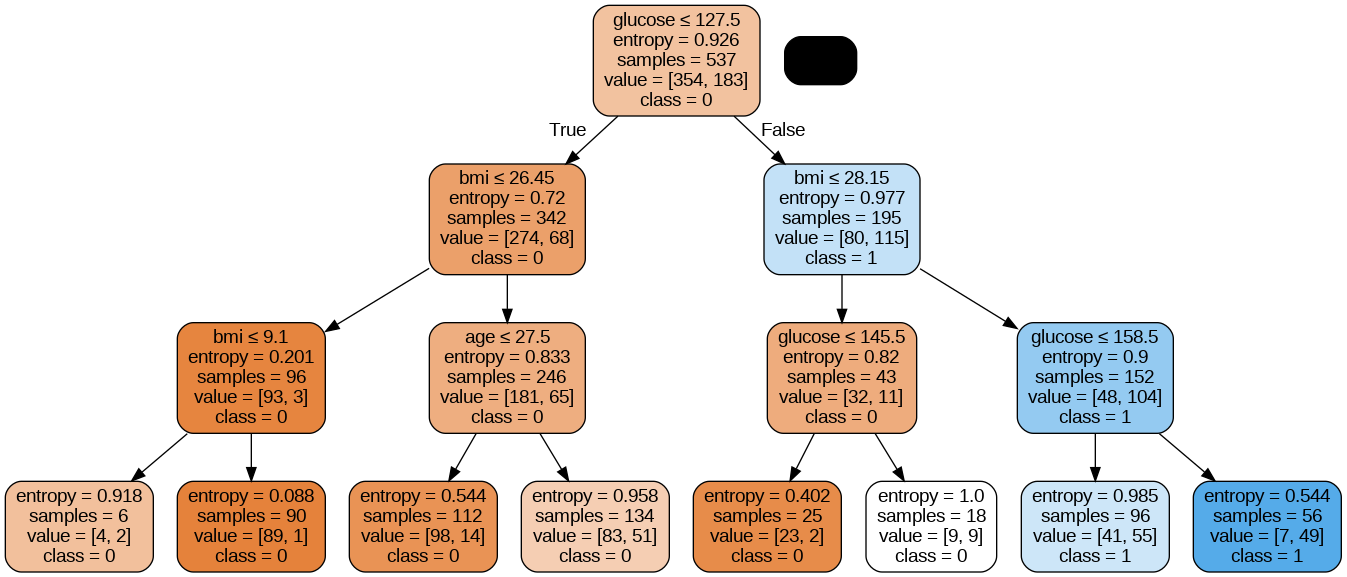
                    filled=True, rounded=True,

                    special\_characters=True, feature\_names = feature\_cols,class\_names=['0','1'])

graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_png('diabetes.png')

Image(graph.create\_png())



**EX-5** Perform classification using Support Vector Machines

**Aim:**

To Perform classification using Support Vector Machines

**Program:**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

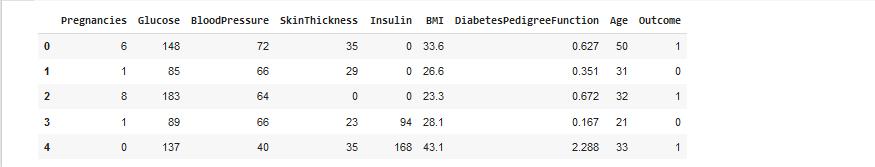
from google.colab import files

uploaded=files.upload()

****

data=pd.read\_csv("diabetes.csv")

data.head()

****

non\_zero = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']

for coloumn in non\_zero:

    data[coloumn] = data[coloumn].replace(0,np.NaN)

    mean = int(data[coloumn].mean(skipna = True))

    data[coloumn] = data[coloumn].replace(np.NaN,mean)

    print(data[coloumn])

from sklearn.model\_selection import train\_test\_split

X =data.iloc[:,0:8]

y =data.iloc[:,8]

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=0, stratify=y)

X.head()

0 148.0

1 85.0

2 183.0

3 89.0

4 137.0

...

763 101.0

764 122.0

765 121.0

766 126.0

767 93.0

Name: Glucose, Length: 768, dtype: float64

0 72.0

1 66.0

2 64.0

3 66.0

4 40.0

...

763 76.0

764 70.0

765 72.0

766 60.0

767 70.0

Name: BloodPressure, Length: 768, dtype: float64

0 35.0

1 29.0

2 29.0

3 23.0

4 35.0

...

763 48.0

764 27.0

765 23.0

766 29.0

767 31.0

Name: SkinThickness, Length: 768, dtype: float64

0 155.0

1 155.0

2 155.0

3 94.0

4 168.0

...

763 180.0

764 155.0

765 112.0

766 155.0

767 155.0

Name: Insulin, Length: 768, dtype: float64

0 33.6

1 26.6

2 23.3

3 28.1

4 43.1

...

763 32.9

764 36.8

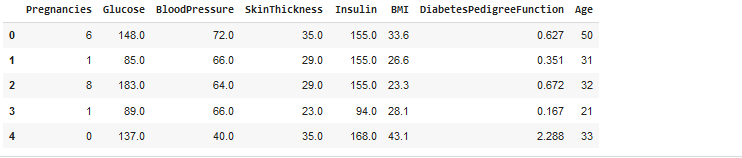
765 26.2

766 30.1

767 30.4

Name: BMI, Length: 768, dtype: float64

|  |  |
| --- | --- |

****

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.transform(X\_test)

from sklearn import svm

svm1 = svm.SVC(kernel='linear', C = 0.01)

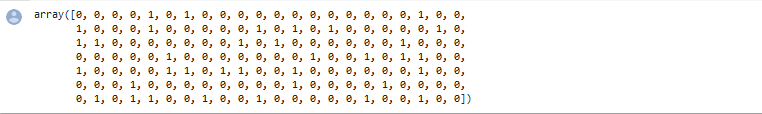
svm1.fit(X\_test,y\_test)

svm.SVC(C=0.01, kernel='linear')

y\_train\_pred = svm1.predict(X\_train)

y\_test\_pred = svm1.predict(X\_test)

y\_test\_pred

**** from sklearn.metrics import accuracy\_score,confusion\_matrix

confusion\_matrix(y\_test,y\_test\_pred)

**** accuracy\_score(y\_test,y\_test\_pred)

****

**Result:**

**EX-6** Perform multivariate classification and regression

**Aim:**

To Perform multivariate classification and regression

**Program:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

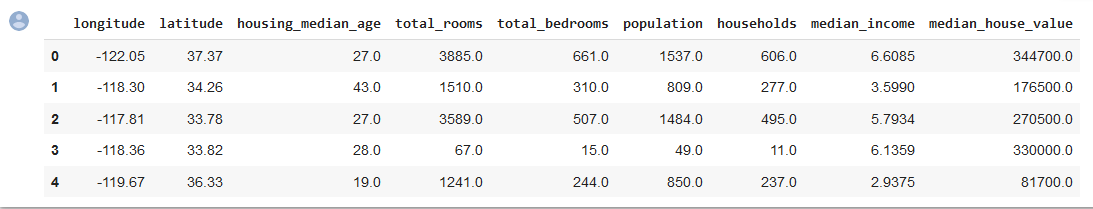
import seaborn as sns

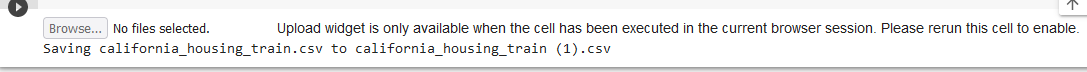
from google.colab import files

upladed=files.upload()

**** test=pd.read\_csv("california\_housing\_test.csv")

test.head()

**** upladed=files.upload()

**** train=pd.read\_csv("california\_housing\_train.csv")

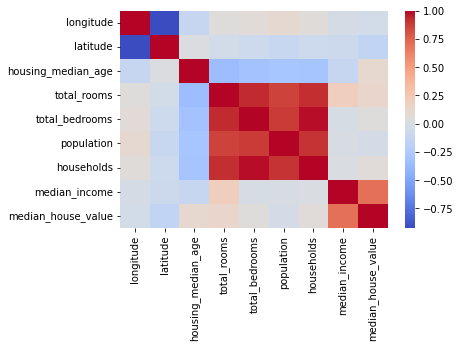
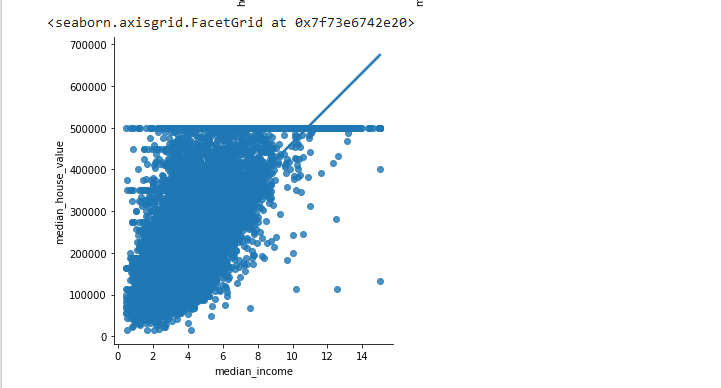
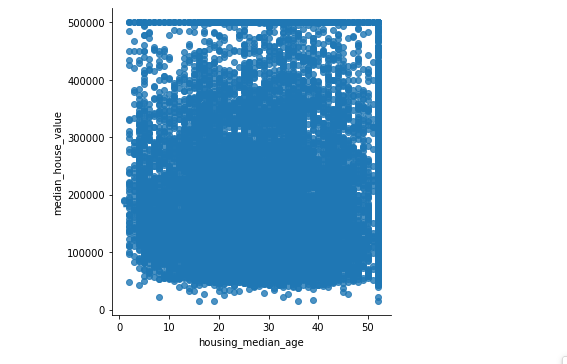
plt.figure()

sns.heatmap(train.corr(),cmap='coolwarm')

plt.show()

sns.lmplot(x='median\_income',y='median\_house\_value',data=train)

sns.lmplot(x='housing\_median\_age',y='median\_house\_value',data=train)

**** ****  data=train

data=data[['total\_rooms','total\_bedrooms','housing\_median\_age','median\_income','population','households']]

data.info()

data['total\_rooms']=data['total\_rooms'].fillna(data['total\_rooms'].mean())

data['total\_bedrooms']=data['total\_bedrooms'].fillna(data['total\_bedrooms'].mean())

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 17000 entries, 0 to 16999

Data columns (total 6 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 total\_rooms 17000 non-null float64

1 total\_bedrooms 17000 non-null float64

2 housing\_median\_age 17000 non-null float64

3 median\_income 17000 non-null float64

4 population 17000 non-null float64

5 households 17000 non-null float64

dtypes: float64(6)

memory usage: 797.0 KB

<ipython-input-8-2fa8111f12dc>:4: SettingWithCopyWarning:

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>

data['total\_rooms']=data['total\_rooms'].fillna(data['total\_rooms'].mean())

<ipython-input-8-2fa8111f12dc>:5: SettingWithCopyWarning:

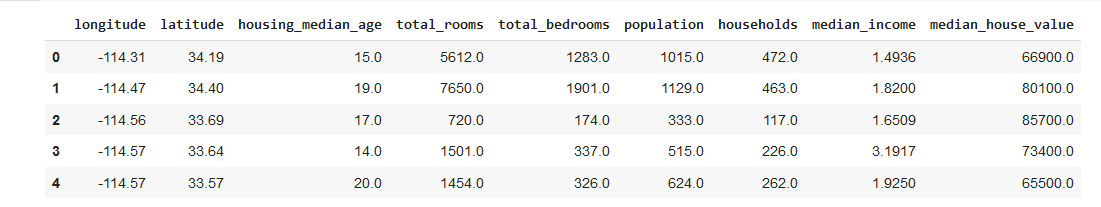
A value is trying to be set on a copy of a slice from a DataFrame.

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>

data['total\_bedrooms']=data['total\_bedrooms'].fillna(data['total\_bedrooms'].mean())

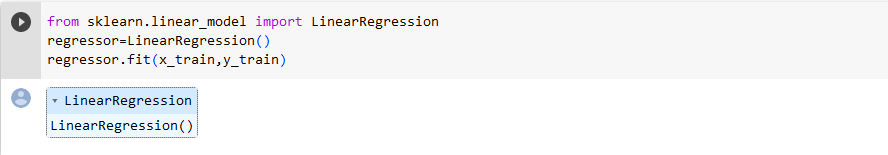
train.head()

**** from sklearn.model\_selection import train\_test\_split

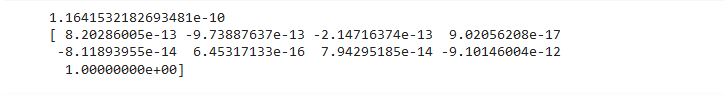
y=train.iloc[:,8]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(train, y,test\_size=0.2,random\_state=0)

print(y.name)

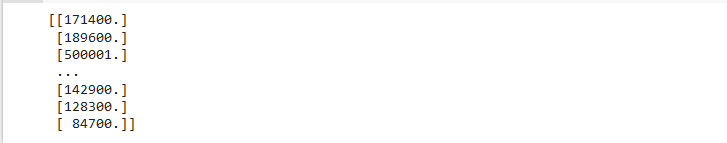
**** **** print(regressor.intercept\_)

print(regressor.coef\_)

**** predictions=regressor.predict(x\_test)

predictions=predictions.reshape(-1,1)

print(predictions)

**** from sklearn.metrics import mean\_squared\_error

print('MSE:',mean\_squared\_error(y\_test,predictions))

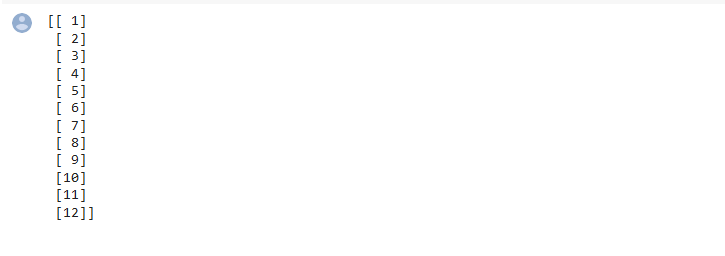
print('RMSE:',np.sqrt(mean\_squared\_error(y\_test,predictions)))

**** import numpy as np

arr=np.array([1,2,3,4,5,6,7,8,9,10,11,12])

newarr=arr.reshape(-1,1)

print(newarr)



**Result:**

**EX-7** Develop a program to implement feed-forward neural networks

**Aim:**

To develop a program to implement feed-forward neural networks

**Program:**

import math

import pandas as pd

from keras import models, layers, optimizers, regularizers

import numpy as np

import random

from sklearn import model\_selection, preprocessing

import tensorflow as tf

from tqdm import tqdm

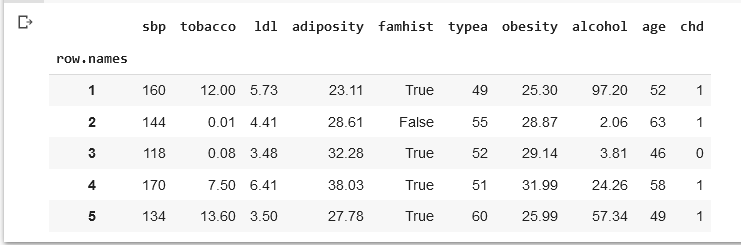
import matplotlib.pyplot as plt

file\_name = '/content/SAheart.data'

data = pd.read\_csv(file\_name, sep=',', index\_col=0)

data['famhist'] = data['famhist'] == 'Present'

data.head()

**** n\_test = int(math.ceil(len(data) \* 0.3))

random.seed(42)

test\_ixs = random.sample(list(range(len(data))), n\_test)

train\_ixs = [ix for ix in range(len(data)) if ix not in test\_ixs]

train = data.iloc[train\_ixs, :]

test = data.iloc[test\_ixs, :]

print(len(train))

print(len(test))

**** #features = ['sbp', 'tobacco', 'ldl', 'adiposity', 'famhist', 'typea', 'obesity', 'alcohol', 'age']

features = ['adiposity', 'age']

response = 'chd'

x\_train = train[features]

y\_train = train[response]

x\_test = test[features]

y\_test = test[response]

x\_train = preprocessing.normalize(x\_train)

x\_test = preprocessing.normalize(x\_test)

hidden\_units = 10     # how many neurons in the hidden layer

activation = 'relu'   # activation function for hidden layer

l2 = 0.01             # regularization - how much we penalize large parameter values

learning\_rate = 0.01  # how big our steps are in gradient descent

epochs = 5            # how many epochs to train for

batch\_size = 16       # how many samples to use for each gradient descent update

# create a sequential model

model = models.Sequential()

# add the hidden layer

model.add(layers.Dense(input\_dim=len(features),

                       units=hidden\_units,

                       activation=activation))

# add the output layer

model.add(layers.Dense(input\_dim=hidden\_units,

                       units=1,

                       activation='sigmoid'))

# define our loss function and optimizer

model.compile(loss='binary\_crossentropy',

              # Adam is a kind of gradient descent

              optimizer=optimizers.Adam(lr=learning\_rate),

              metrics=['accuracy'])

/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117: UserWarning: The `lr` argument is deprecated, use `learning\_rate` instead.

super().\_\_init\_\_(name, \*\*kwargs)

# train the parameters

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=batch\_size)

# evaluate accuracy

train\_acc = model.evaluate(x\_train, y\_train, batch\_size=32)[1]

test\_acc = model.evaluate(x\_test, y\_test, batch\_size=32)[1]

print('Training accuracy: %s' % train\_acc)

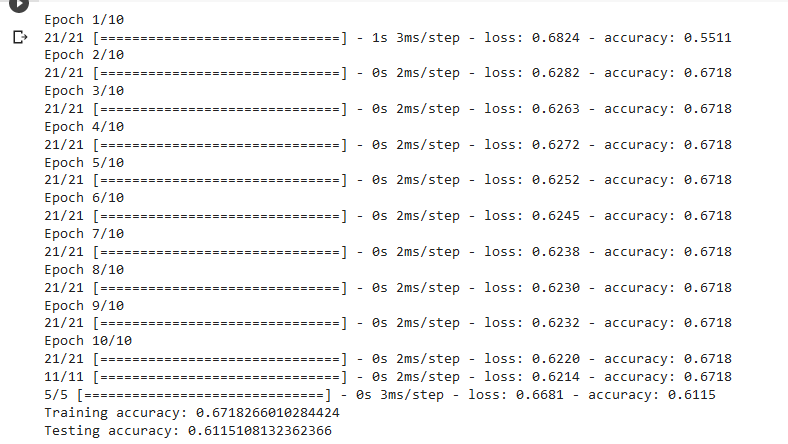
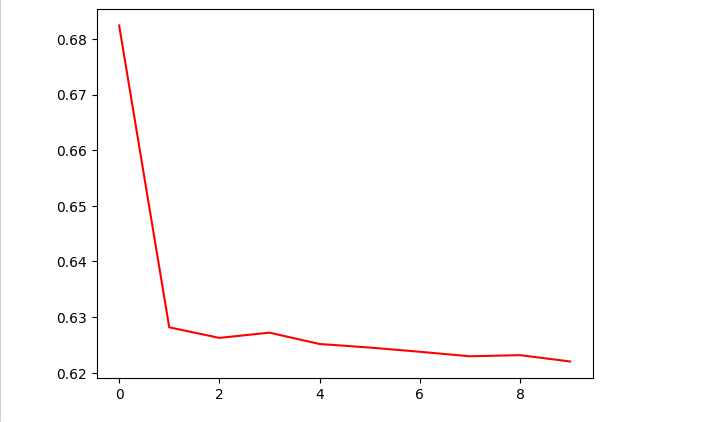
print('Testing accuracy: %s' % test\_acc)

losses = history.history['loss']

plt.plot(range(len(losses)), losses, 'r')

plt.show()

### RUN IT AGAIN! ###

**** ****

**Result:**

**EX-8** Implement K-means clustering

**Aim:**

To Implement K-means clustering

**Program:**

import numpy as np

import pandas as pd

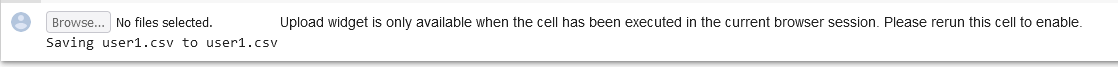
import matplotlib.pyplot as plt

from sklearn.cluster import  KMeans

import sklearn.metrics as metrics

from google.colab import files

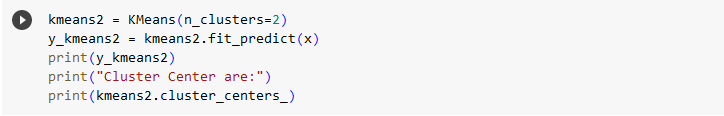
uploaded = files.upload()

**** df = pd.read\_csv("/content/user1.csv")

df

 x=df.iloc[:,[0,1]].values

print(x)

 ****

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

Cluster Center are:

[[0.488 0.29 ]

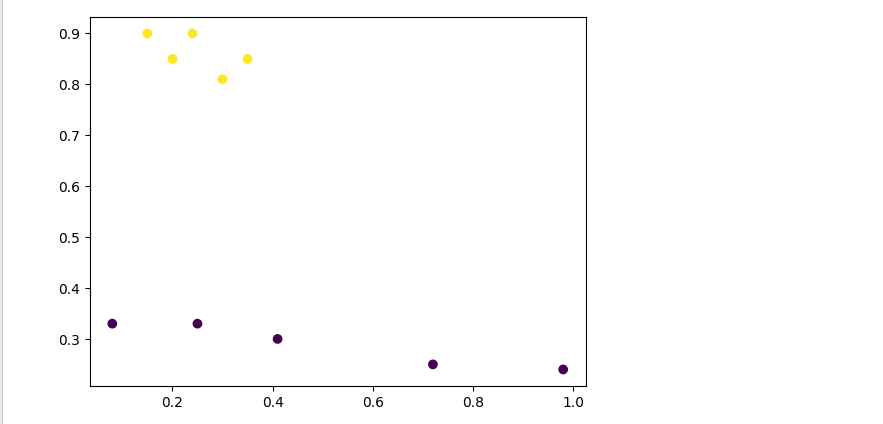
[0.248 0.862]]

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/\_kmeans.py:870: FutureWarning: The default value of `n\_init` will change from 10 to 'auto' in 1.4. Set the value of `n\_init` explicitly to suppress the warning

warnings.warn(

plt.scatter(x[:,0],x[:,1],c=y\_kmeans2,cmap='viridis')

plt.show()

****

**Result:**

**EX-9** Develop a simple application to demonstrate reinforcement learning

**Aim:**

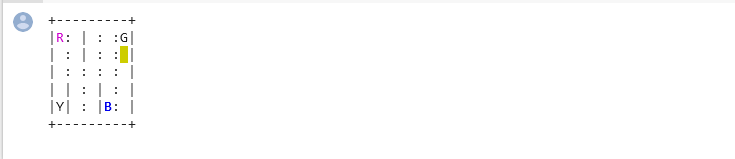
To develop a simple application to demonstrate reinforcement learning

**Program:**

import gym

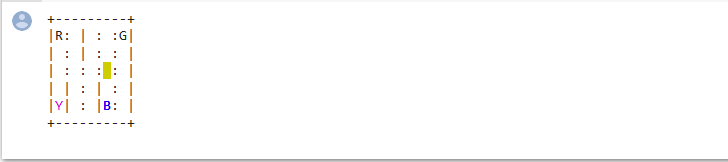
env = gym.make("Taxi-v3").env

env.render()

****

env.reset() # reset environment to a new, random state

env.render()

****

print("Action Space {}".format(env.action\_space))

print("State Space {}".format(env.observation\_space))

****

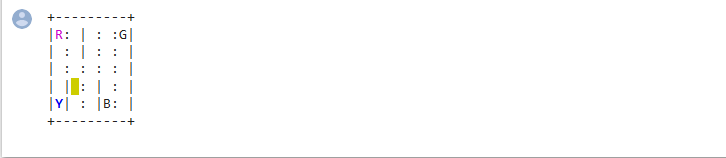
state = env.encode(3, 1, 2, 0) # (taxi row, taxi column, passenger index, destination index)

print("State:", state)

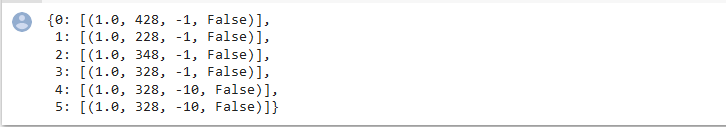
****

env.s = state

env.render()

****

env.P[328]

****

env.s = 328  # set environment to illustration's state

epochs = 0

penalties, reward = 0, 0

frames = [] # for animation

done = False

while not done:

    action = env.action\_space.sample()

    state, reward, done, info = env.step(action)

    if reward == -10:

        penalties += 1

    # Put each rendered frame into dict for animation

    frames.append({

        'frame': env.render(mode='ansi'),

        'state': state,

        'action': action,

        'reward': reward

        }

    )

    epochs += 1

print("Timesteps taken: {}".format(epochs))

print("Penalties incurred: {}".format(penalties))

****

def print\_frames(frames):

    for i, frame in enumerate(frames):

        clear\_output(wait=True)

        print(frame['frame'])

        print(f"Timestep: {i + 1}")

        print(f"State: {frame['state']}")

        print(f"Action: {frame['action']}")

        print(f"Reward: {frame['reward']}")

        sleep(.1)

print\_frames(frames)

****

**Result:**

**EX-10** Assess machine learning algorithms using cross validation methods

**Aim:**

To assess machine learning algorithms using cross validation methods

**Program:**

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report

from sklearn import datasets

from sklearn import svm

X, y = datasets.load\_iris(return\_X\_y=True)

X.shape, y.shape

****

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

  X, y, test\_size=0.3, random\_state=0)

X\_train.shape, y\_train.shape

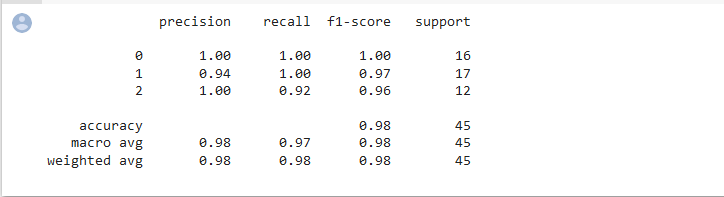
**** X\_test.shape, y\_test.shape

**** clf = svm.SVC(kernel='linear', C=1)

clf.fit(X\_train, y\_train)

**** Y\_predict = clf.predict(X\_test)

print(classification\_report(Y\_predict,y\_test))

**** from sklearn.model\_selection import cross\_val\_score

clf = svm.SVC(kernel='linear', C=1, random\_state=42)

scores = cross\_val\_score(clf, X, y, cv=10)

scores

**** print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(), scores.std()))

****

**Result:**