Problem Statement

The purpose is to predict whether the Pima Indian women shows signs of diabetes or not. We are using a datas
Digestive and Kidney Diseases" which consists of a number of attributes which would help us to perform this pı

Constraints on data collection

All patients whose data has been collected are females at least 21 years old of Pima Indian heritage

```
#Import all the necessary modules
import numpy as np
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

```
df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/pima-indians-diabetes+(3).csv")
df.head()
```

```
   Preg  Plas  Pres  skin  test  mass   pedi  age  class
0     6   148    72    35     0  33.6  0.627   50      1
1     1    85    66    29     0  26.6  0.351   31      0
2     8   183    64     0     0  23.3  0.672   32      1
3     1    89    66    23    94  28.1  0.167   21      0
4     0   137    40    35   168  43.1  2.288   33      1
```

Q1. Load the PIMA Indian Diabetes file into Python Dat

The file can be accessed directly from the URL (https://archive.ics.uci.edu/ml/machine-learning-databases/pim
you may first download it to a local folder and then load it into Python dataframe. Let us assume the data frame

```
pima_df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/pima-indians-diabetes+(3).csv")
pima_df.head()
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0

It is always a good practice to eye-ball raw data to get a feel of the data in terms of number of structure of the fi
general idea of likely challenges in the dataset. You would notice that it is a comma separated file. There are no
find out about each attribute the name. What information is available about the data.

Q2. Print 10 samples from the dataset

```
pima_df.head(10)
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

Q3 Print the datatypes of each column and the shape o

```
pima_df.dtypes
```

```
Preg        int64
Plas        int64
Pres        int64
skin        int64
test        int64
mass      float64
```

```
pima_df.shape
```

```
(768, 9)
```

```
pima_df.sample(n=10)
print(pima_df.dtypes)
print(pima_df.shape)
```

```
Preg        int64
Plas        int64
Pres        int64
skin        int64
test        int64
mass      float64
pedi      float64
age         int64
class       int64
dtype: object
(768, 9)
```

```
pima_df.sample(n=10)
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
330	8	118	72	19	0	23.1	1.476	46	0
658	11	127	106	0	0	39.0	0.190	51	0
235	4	171	72	0	0	43.6	0.479	26	1
128	1	117	88	24	145	34.5	0.403	40	1
262	4	95	70	32	0	32.1	0.612	24	0
299	8	112	72	0	0	23.6	0.840	58	0
583	8	100	76	0	0	38.7	0.190	42	0
644	3	103	72	30	152	27.6	0.730	27	0
545	8	186	90	35	225	34.5	0.423	37	1
45	0	180	66	39	0	42.0	1.893	25	1

There are '0's in the data. Are they really valid '0's or they are missing values? Plasma, BP, skin thickness etc. the
logically to understand this.

Q4 Replace all the 0s in the column with the median of accordingly.

```
pima_df.loc[pima_df.Plas == 0, 'Plas'] = pima_df.Plas.median()

pima_df.loc[pima_df.Pres == 0, 'Pres'] = pima_df.Pres.median()

pima_df.loc[pima_df.skin == 0, 'skin'] = pima_df.skin.median()

pima_df.loc[pima_df.test == 0, 'test'] = pima_df.test.median()

pima_df.loc[pima_df.mass == 0, 'mass'] = pima_df.mass.median()

pima_df.sample(n=10)
```

	Preg	Plas	Pres	skin	test	mass	pedi	age	class
673	3	123.0	100.0	35.0	240.0	57.3	0.880	22	0
485	0	135.0	68.0	42.0	250.0	42.3	0.365	24	1
47	2	71.0	70.0	27.0	30.5	28.0	0.586	22	0
670	6	165.0	68.0	26.0	168.0	33.6	0.631	49	0
95	6	144.0	72.0	27.0	228.0	33.9	0.255	40	0
270	10	101.0	86.0	37.0	30.5	45.6	1.136	38	1
663	9	145.0	80.0	46.0	130.0	37.9	0.637	40	1
443	8	108.0	70.0	23.0	30.5	30.5	0.955	33	1
586	8	143.0	66.0	23.0	30.5	34.9	0.129	41	1
379	0	93.0	100.0	39.0	72.0	43.4	1.021	35	0

```
pima_df.replace(to_replace={'Plas':0,'Pres':0,'skin':0,'test':0,'mass':'0'},value={'Plas':pima_d
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/core/nanops.py in f(values, axis, skipr
    126                 else:
--> 127                     result = alt(values, axis=axis, skipna=skipna, **kwds)
    128             except Exception:

                                    6 frames
TypeError: float() argument must be a string or a number, not 'method'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent call last)
/usr/local/lib/python3.6/dist-packages/pandas/core/nanops.py in nanmedian(values, ax:
    525     values, mask, dtype, dtype_max, _ = _get_values(values, skipna, mask=masl
    526     if not is_float_dtype(values):
--> 527         values = values.astype('f8')
    528         values[mask] = np.nan
    529
```

Q5 Print the descriptive statistics of each & every colu function

```
pima_df.describe()
```

	Preg	Plas	Pres	skin	test	mass	pe
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.0000
mean	3.845052	121.656250	72.386719	27.334635	94.652344	32.450911	0.4718
std	3.369578	30.438286	12.096642	9.229014	105.547598	6.875366	0.3313
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.200000	0.0780
25%	1.000000	99.750000	64.000000	23.000000	30.500000	27.500000	0.2437
50%	3.000000	117.000000	72.000000	23.000000	31.250000	32.000000	0.3725
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.6262
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.4200

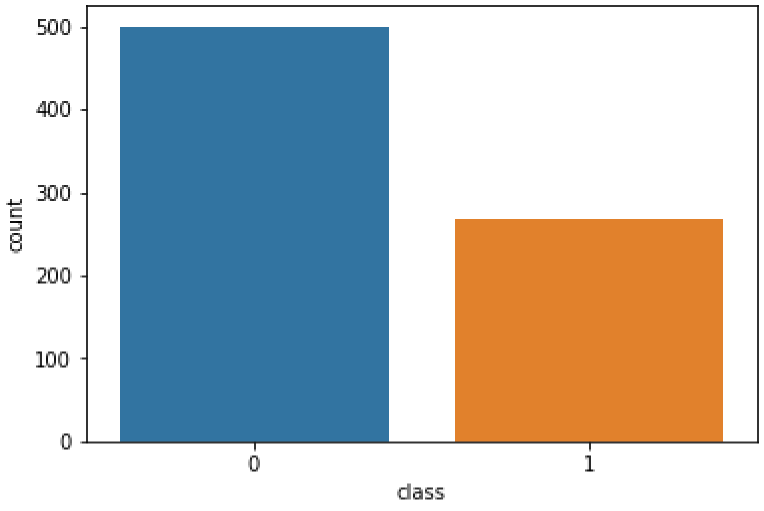
Q6 See the distribution of 'Class' variable and plot it usi

```
pima_df.describe().transpose()
```

	count	mean	std	min	25%	50%	75%	max
Preg	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Plas	768.0	121.656250	30.438286	44.000	99.75000	117.0000	140.25000	199.00
Pres	768.0	72.386719	12.096642	24.000	64.00000	72.0000	80.00000	122.00
skin	768.0	27.334635	9.229014	7.000	23.00000	23.0000	32.00000	99.00
test	768.0	94.652344	105.547598	14.000	30.50000	31.2500	127.25000	846.00
mass	768.0	32.450911	6.875366	18.200	27.50000	32.0000	36.60000	67.10
pedi	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00

```
sns.countplot(x='class', data=pima_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4e64f96e10>
```



Just for your understanding - Using univariate analysis attributes for their basic statistic such as central value are your observations (any two attributes). Its an optio graded.

Q7. Use pairplots and correlation method to observe th different variables and state your insights.

```
pima_df.corr()
```

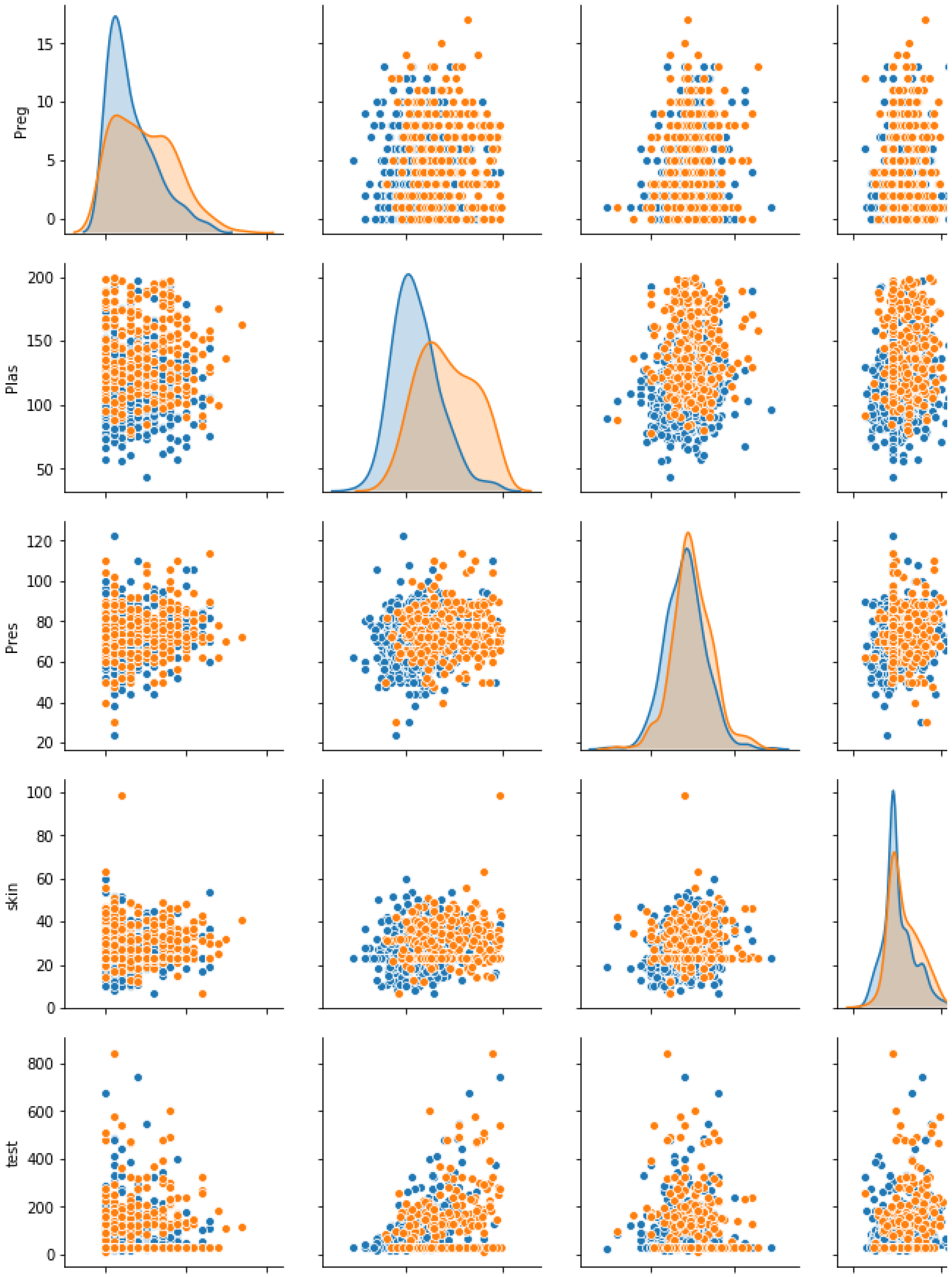
	Preg	Plas	Pres	skin	test	mass	pedi	age
Preg	1.000000	0.128213	0.208615	0.032568	-0.055697	0.021546	-0.033523	0.544341
Plas	0.128213	1.000000	0.218937	0.172143	0.357573	0.231400	0.137327	0.26690
Pres	0.208615	0.218937	1.000000	0.147809	-0.028721	0.281132	-0.002378	0.32491
skin	0.032568	0.172143	0.147809	1.000000	0.238188	0.546951	0.142977	0.05451
test	-0.055697	0.357573	-0.028721	0.238188	1.000000	0.189022	0.178029	-0.01541
mass	0.021546	0.231400	0.281132	0.546951	0.189022	1.000000	0.153506	0.02574
pedi	-0.033523	0.137327	-0.002378	0.142977	0.178029	0.153506	1.000000	0.033561
age	0.544341	0.266909	0.324915	0.054514	-0.015413	0.025744	0.033561	1.00000
class	0.221898	0.492782	0.165723	0.189065	0.148457	0.312249	0.173844	0.23835

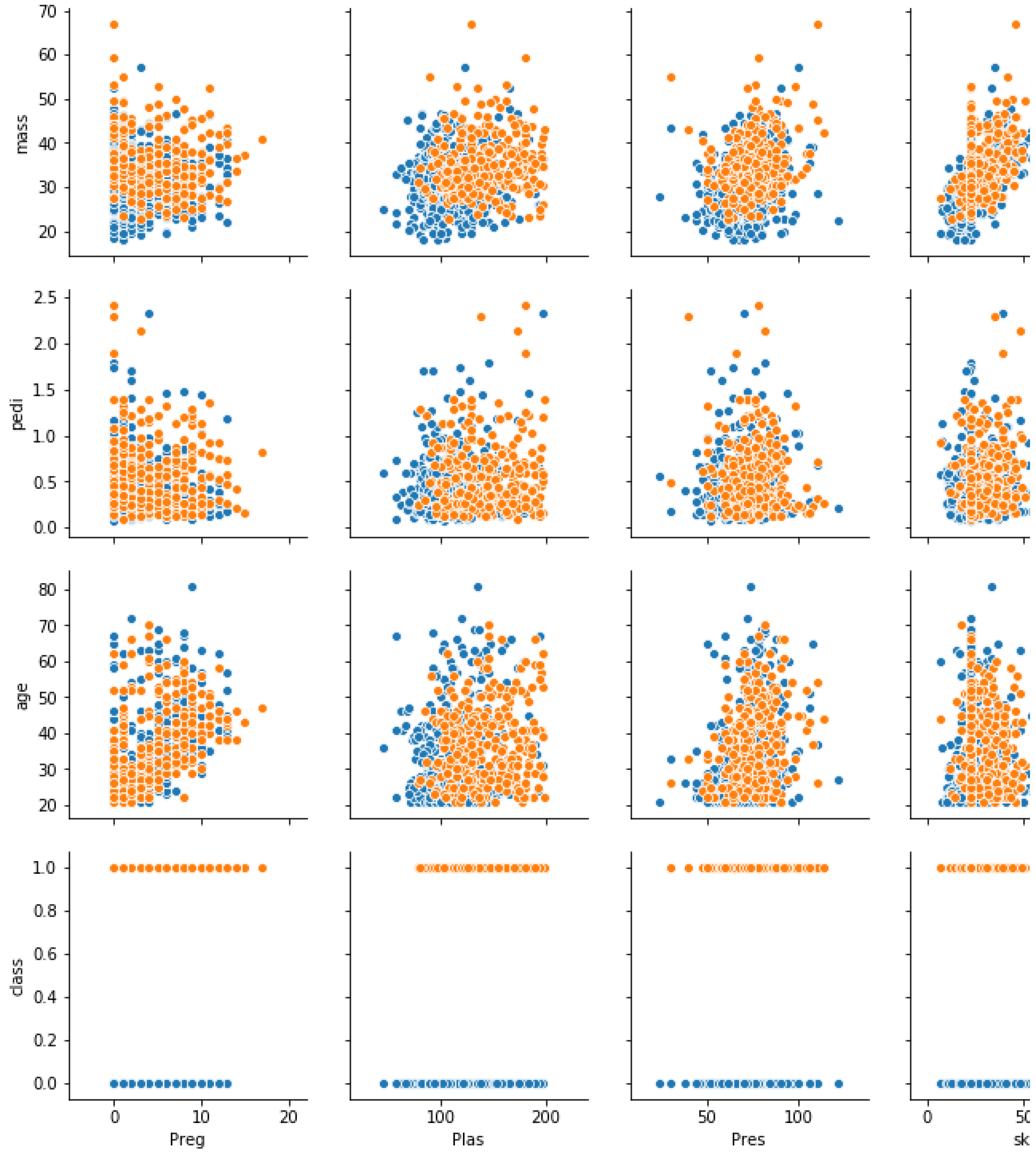
Using the plot - infer the relationship between different variables

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from IPython.display import Image
```

```
sns.pairplot(pima_df,hue='class')
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:487: RuntimeW
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34: Runt
  FAC1 = 2*(np.pi*bw/RANGE)**2
<seaborn.axisgrid.PairGrid at 0x7f4e64e58358>
```





Q8 Split the pima_df into training and test set in the rat (Training:Test).

```
x=pima_df.iloc[:,:8]
y=pima_df.iloc[:,8]
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=0)
xtrain.shape,xtest.shape
```

```
((537, 8), (231, 8))
```

Q9 Create the decision tree model using “entropy” met entropy and fit it to training data.

```
dt_model = DecisionTreeClassifier(criterion = 'entropy' )
dt_model.fit(xtrain, ytrain)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=None, splitter='best')
```

Q10 Print the accuracy of the model & print the confusi

```
dt_model.score(xtest , ytest)
```

```
0.7186147186147186
```

Print the feature importance of the decision model - Optional

```
y_predict = dt_model.predict(xtest)
```

```
y_predict
```

```
array([1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1])
```

```
print(metrics.confusion_matrix(ytest,y_predict),)
```