1.

Given the iris dataset with 4 features and 1 target columns, we have to apply PCA to reduce the dimensionality .

a. Standardize the data :

Firstly we will divide the dataset into X and Y , where X consists of feature columns and Y consists of target column .

To standardize the data :

$$X_{changed} = \frac{X - \mu}{\sigma}$$

We will subtract mean of the column from each value in column and divide it by the standard deviation of the column .

The code used to do this :

```
for column in X:
  mean_col = X[column].mean()
  std_dev= X[column].std()
  i=0
  for row in X[column]:
    row= (row-mean_col)/std_dev
    X[column][i]=row
    i+=1
```

b. How many eigenvectors are required to preserve at least 90% of the data variation?

To implement PCA , we need a covariance matrix , which we can calculate using

```
 features = X_ar

cov_matrix = np.cov(features, rowvar=False)
```

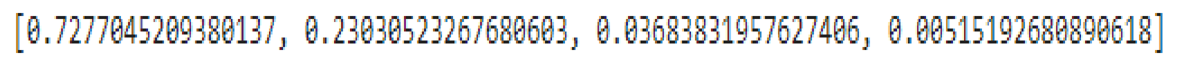
np.cov takes each row of the input as a variable, with the columns representing different values of those variables. To reverse this behavior, pass rowvar=False.

To find the eigen vectors and eigen values of the covariance matrix , we use

```
values, vectors = np.linalg.eig(cov_matrix)

values
```

Higher the eigen value, higher is the explained variance , to calculate variance of dataset for each eigen vector, we use :

```
variances = []

for i in range(len(values)):

    variances.append(values[i] / np.sum(values))
```

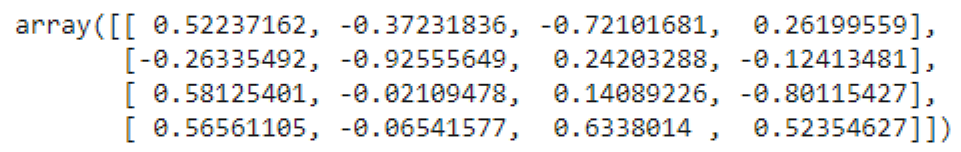


```
[0.7277045209380137, 0.23030523267680603, 0.03683831957627406, 0.005151926808906018]
```

From variance values corresponding to each eigen vector, we realised that if we take **1st 2 eigen vectors**, it will preserve approximately 95% of data variation .

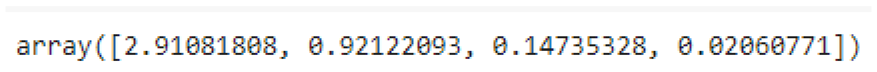
c. Look at the first eigenvector. Dimensions are the primary contributors :

Eigen vectors found in (b.) :



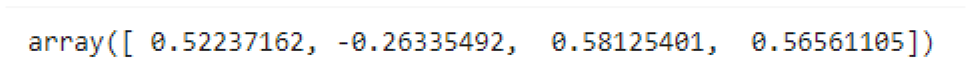
```
array([[ 0.52237162, -0.37231836, -0.72101681,  0.26199559],
       [-0.26335492, -0.92555649,  0.24203288, -0.12413481],
       [ 0.58125401, -0.02109478,  0.14089226, -0.80115427],
       [ 0.56561105, -0.06541577,  0.6338014 ,  0.52354627]])
```

Eigen values found in (b.) :



```
array([2.91081808, 0.92122093, 0.14735328, 0.02060771])
```

The first eigen vector corresponds to the first column of eigen vectors array which is:



```
array([ 0.52237162, -0.26335492,  0.58125401,  0.56561105])
```

We can see that dimensions corresponding to the coefficients `0.58125401` and `0.56561105` in 1st eigen vector are the primary contributors .

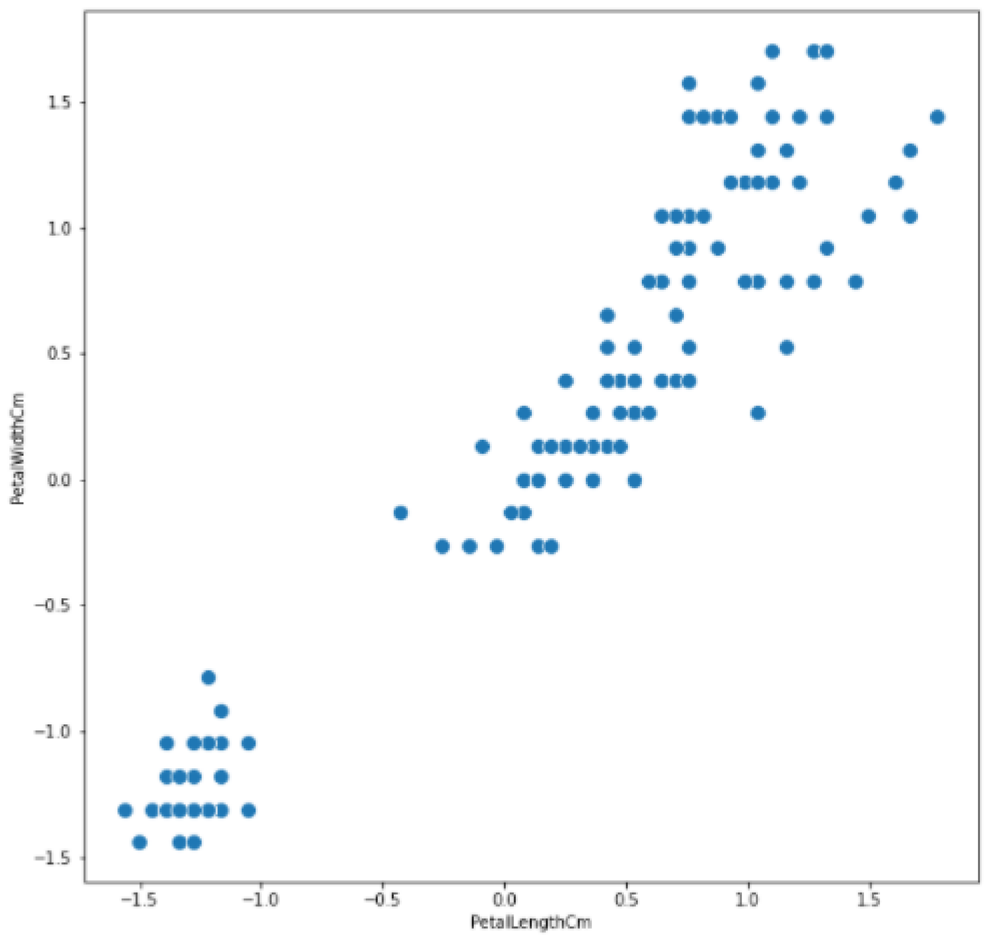
To find the correlation between corresponding features :

```
correlation = X['PetalLengthCm'].corr(X['PetalWidthCm'])
```

```
The correlation comes out to

0.9627570970509667
```

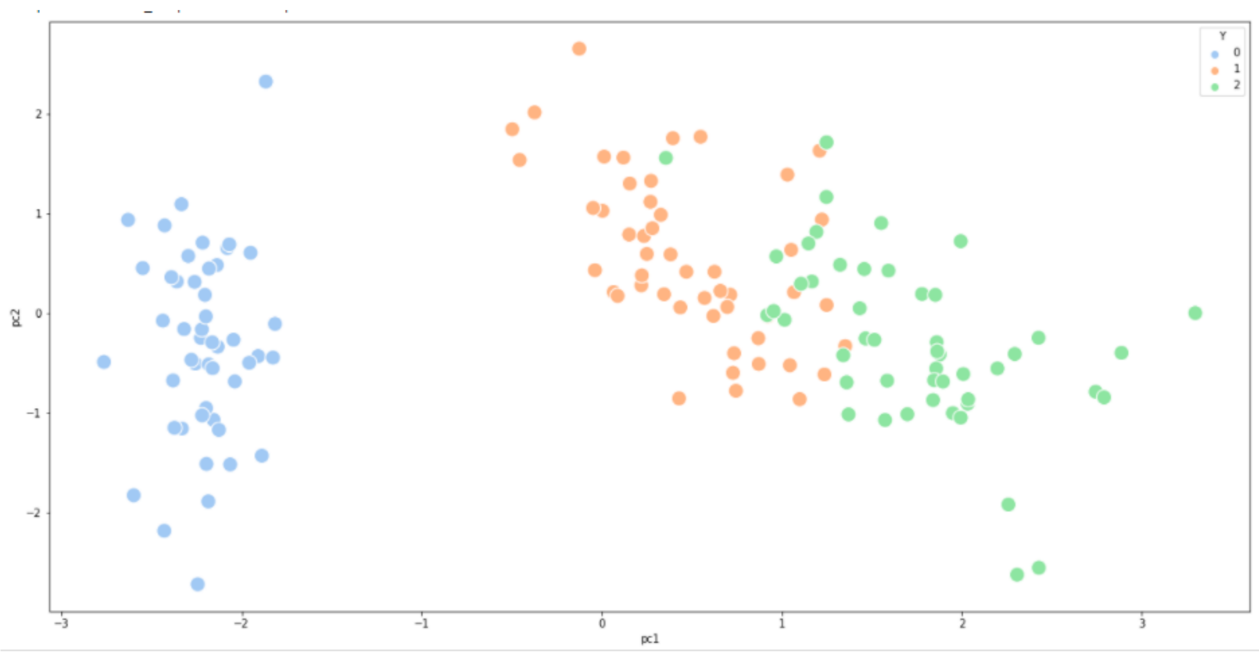
Which is a **large positive correlation** .

The plot between the variables comes out to be:



which apparently proves that the two variables are highly correlated

d.   **Show a plot of your transformed data using the first two eigenvectors.**



Plot of transformed data using PCA

0--- Iris-setosa

1--- Iris-virginica

2--- Iris-versicolor

2. On the same Iris dataset , we are required to implement lda and compare the results of lda and pca .

a. Compare the results of PCA and LDA

To implement LDA on iris dataset, we use sklearn’s `LinearDiscriminantAnalysis` .

```
from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis

clf = LinearDiscriminantAnalysis(n_components=2 ,
solver='eigen')
```

As we have to compare results of PCA and LDA , we want same number of components . From 1st question, we got 2 principal components , so here also we will put `n_components=2` .Similiarily `solver='eigen'`.

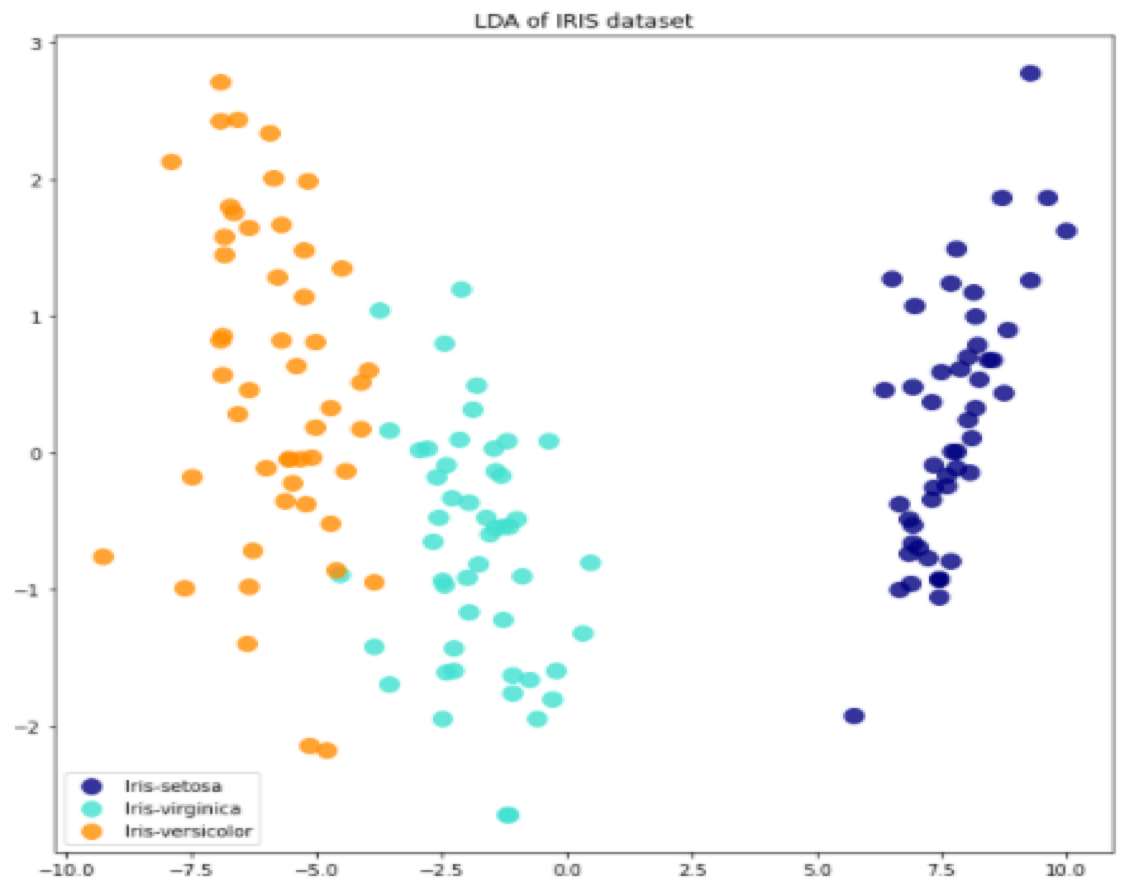
```
X_lda =clf.fit(X, Y).transform(X)
```

	Explained variance ratio
LDA	[0.99147248 0.00852752]

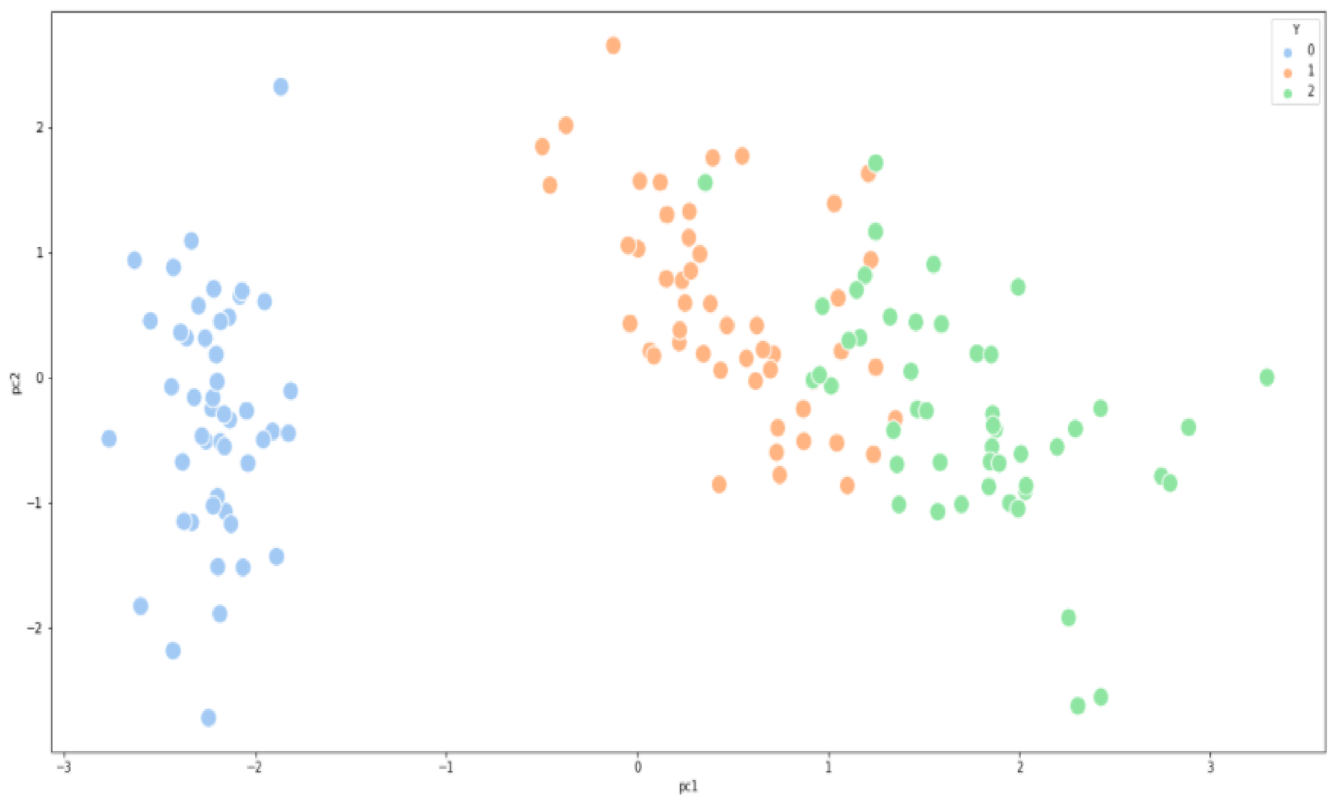
PCA	`[0.727705 0.230305]`

b. Plot the distribution of samples using the first 2 principal components and the first 2 linear discriminants.

Using first 2 linear discriminants:



Using first 2 principal components:



0--- Iris-setosa

1--- Iris-virginica

2--- Iris-versicolor

c. Learn a Bayes classifier using the original features and compare its performance with the features obtained

Using original features :

```
clf_nb = GaussianNB()

clf_nb.fit(X_train, y_train)
```

Using Features obtained after performing LDA :

```
clf_com = LinearDiscriminantAnalysis(n_components=2 ,
solver='eigen')

X_lda_com =clf_com.fit(X_train, y_train).transform(X)
```

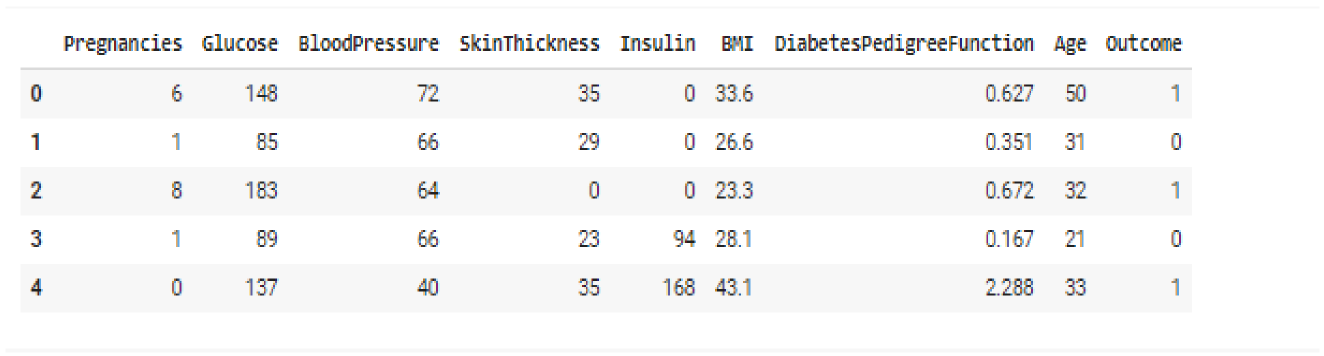
	Accuracy score
Using all features	0.96
Features obtained after applying LDA	0.98

3.

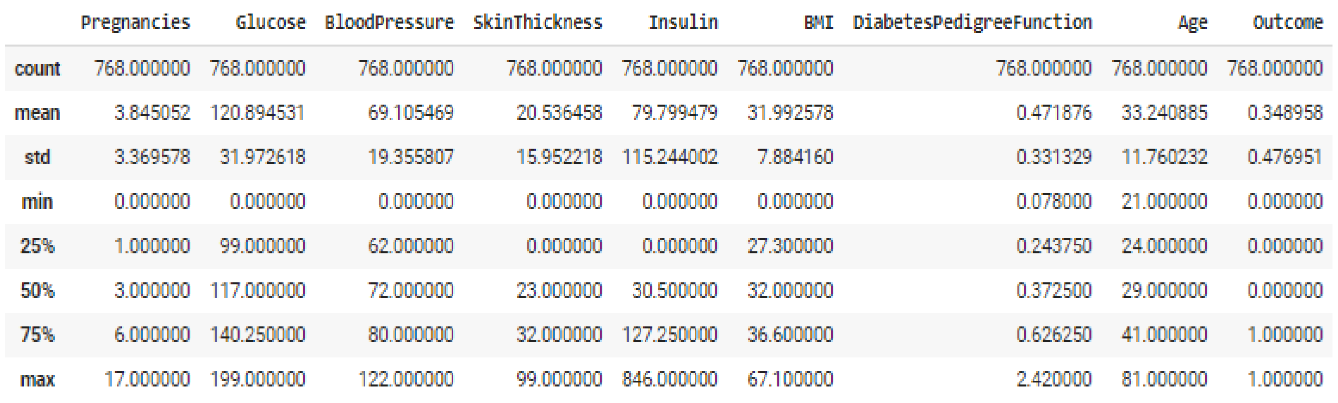
Given the diabetes dataset, which has 8 features and 768 data samples , we need to do feature selection using any 2 techniques taught in the class.

a. Preprocess the data and perform exploratory data analysis:

To explore the data we use various functions like .head(), .shape, .describe() etc .



	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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



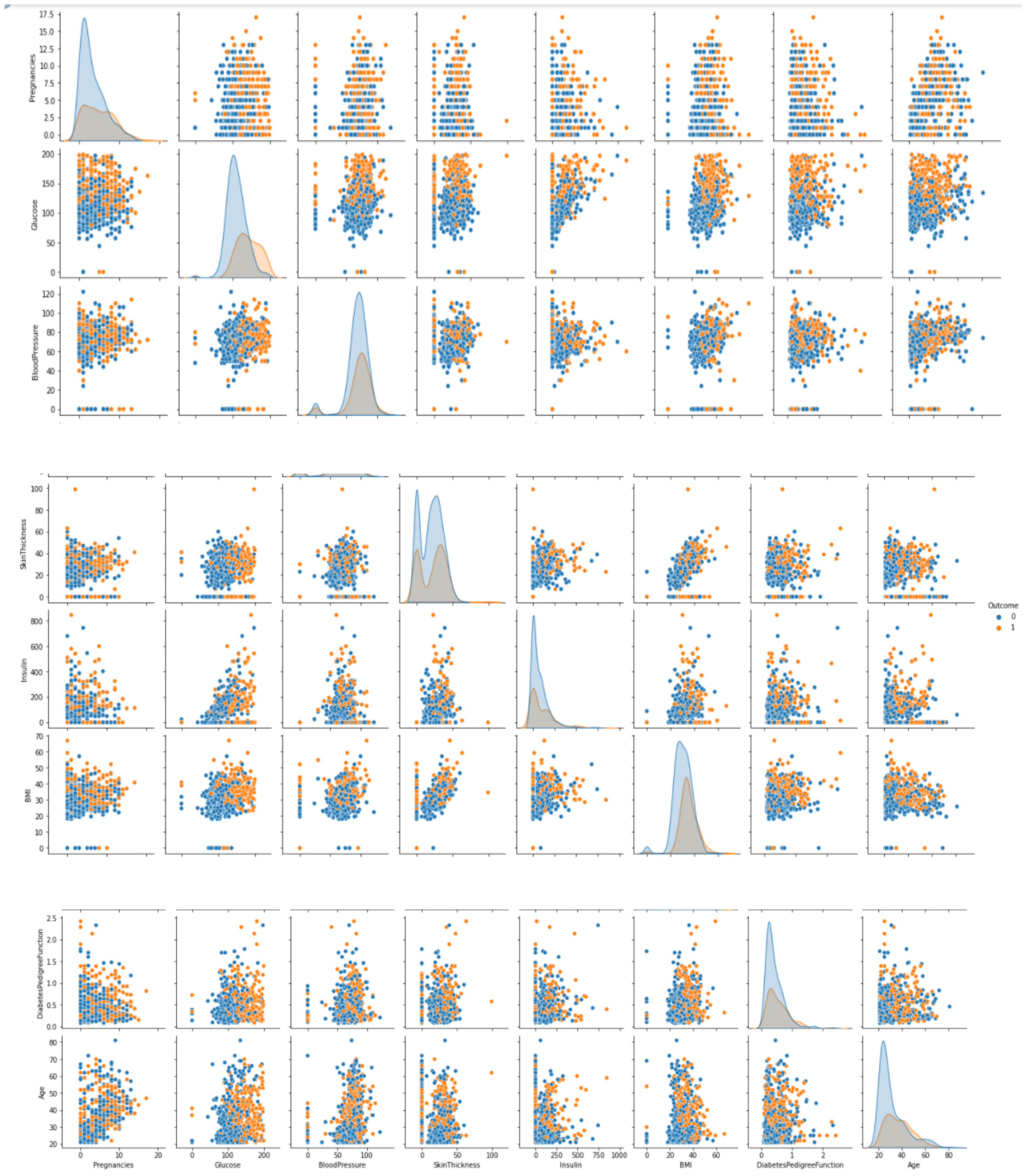
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471876	33.240885	0.348958
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329	11.760232	0.476951
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750	24.000000	0.000000
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500	29.000000	0.000000
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250	41.000000	1.000000
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

Shape of dataset :

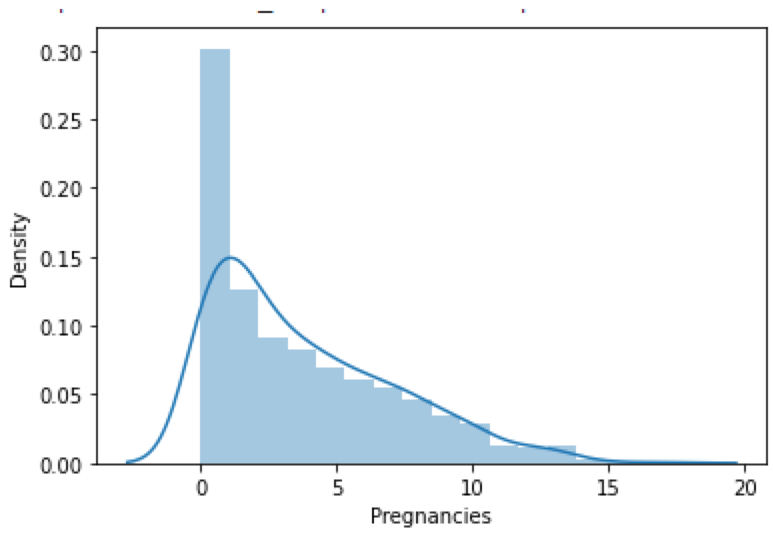
```
768, 9
```

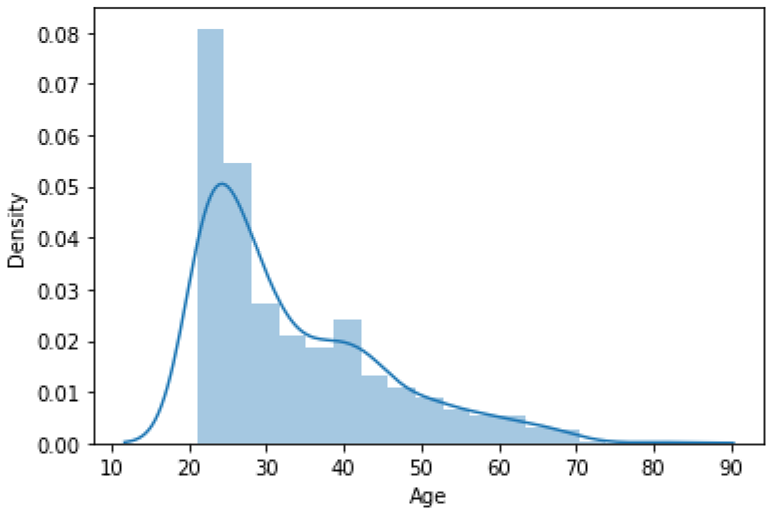
We use pairplot from seaborn library to plot distribution for every pair of variables . As there are 8 features 8 * 8 plots will be created

```
sns.pairplot(diab,hue='Outcome')
```


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
0
1

Clear picture can be seen in the notebook





a. Identify the features having high significance using both of the methods:

We use RFE(Recursive feature elimination) and Forward selection methods to select features

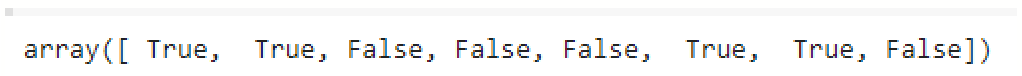
Using RFE :

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
clf_diab = LogisticRegression(random_state=0)

selector = RFE(clf_diab, n_features_to_select=4, step=1)
selector = selector.fit(X_train_diab, y_train_diab)
```

To find which features are selected we can use :

```
selector.support_
```

The output of this function is :



```
array([ True,  True, False, False, False,  True,  True, False])
```

Where True means that the feature corresponding to that index will be selected and False means that that the feature will not be selected

Using Forward Feature Selection:

The below code will give a boolean array similar to that obtained in previous method

```
from sklearn.feature_selection import
SequentialFeatureSelector
sfs = SequentialFeatureSelector(clf_diab,
n_features_to_select=4)
sfs.fit(X_train_diab, y_train_diab)
sfs.support_
```

The output is :

```
array([False,  True, False, False,  True,  True, False,  True])
```

	Features selected
Using REF	**`Pregnancies,Glucose,BMI,DiabetesPedigreeFunction`**
Using forward feature selection	**`Glucose, Insulin, BMI, Age`**

c . Calculate and compare the accuracy and F1 score :

Accuracy and F1 scores can be calculated using the code in the notebook

	Accuracy	F1 score
Using feature selection	`0.7440944881889764`	`0.7082781164413817`
Without using feature selection	`0.7480314960629921`	`0.7187153931339978`

From the table, it is apparent that accuracy and f1 score values for both cases are almost the same even after removing half the number of features

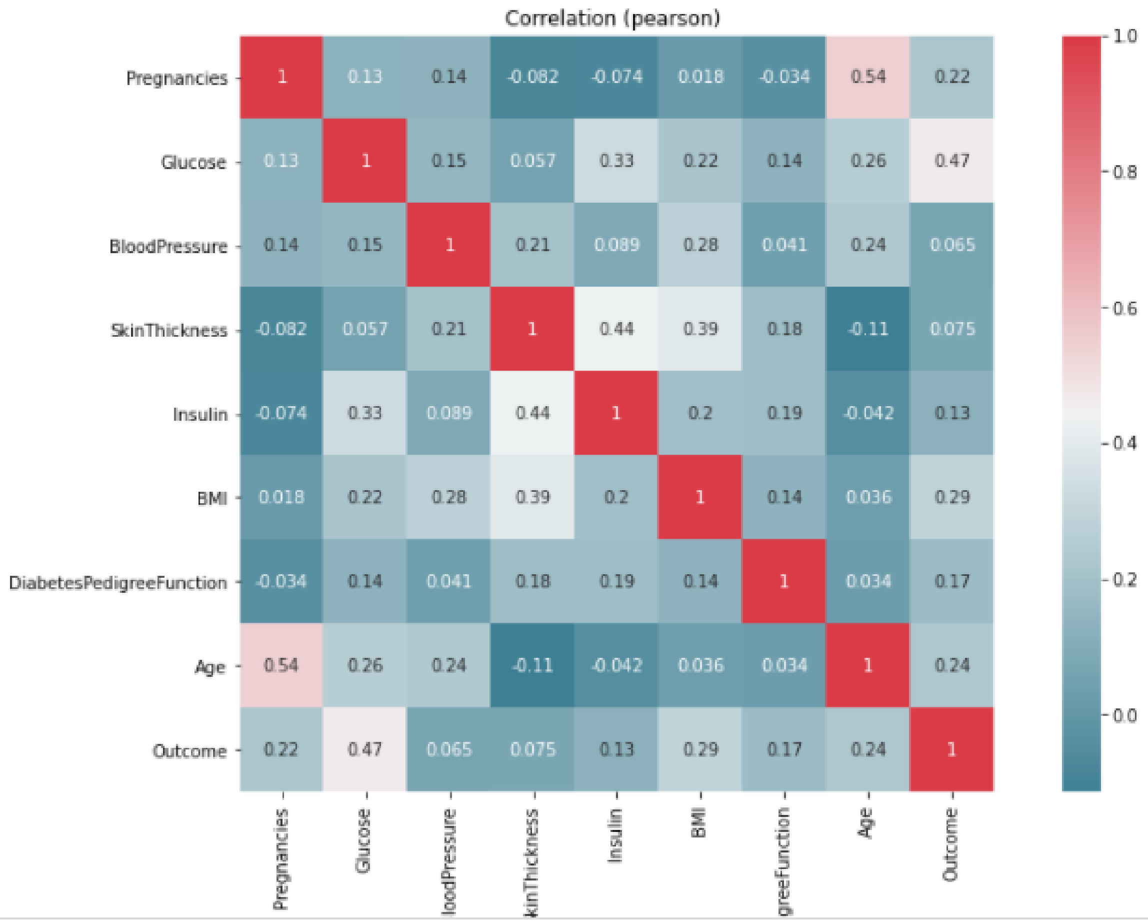
d. Use Pearson Correlation and compute correlated features with a threshold of 70%. :

We can find the correlation between various features in the dataset using :

```
corr_p = diab.corr(method='pearson')
```

A heatmap can be drawn using :

```
ax = sns.heatmap(corr_p, mask=np.zeros_like(corr_p,
dtype=np.bool), cmap=sns.diverging_palette(220, 10,
as_cmap=True),square=True, ax=ax, annot=True)
ax.set_title("Correlation (pearson) ")
plt.show()
```



From the heat map, it is apparent that there are no distinct features which are having a correlation of more than 70% i.e only feature with itself is having a correlation greater than 70% which is 1 obviously which can be shown across the diagonal of the heatmap.