In [1]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
data=pd.read_csv('diabetes.csv')
data
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
0	2	138	62	35	0	33.6	
1	0	84	82	31	125	38.2	
2	0	145	0	0	0	44.2	
3	0	135	68	42	250	42.3	
4	1	139	62	41	480	40.7	
...	...	...	...	...	...	...	
1995	2	75	64	24	55	29.7	
1996	8	179	72	42	130	32.7	
1997	6	85	78	0	0	31.2	
1998	0	129	110	46	130	67.1	
1999	2	81	72	15	76	30.1	

2000 rows × 9 columns

In [2]:

```
data.shape
```

Out[2]:

```
(2000, 9)
```

In [3]:

```
data.head(10)
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	2	138	62	35	0	33.6	0.
1	0	84	82	31	125	38.2	0.
2	0	145	0	0	0	44.2	0.
3	0	135	68	42	250	42.3	0.
4	1	139	62	41	480	40.7	0.
5	0	173	78	32	265	46.5	1.
6	4	99	72	17	0	25.6	0.
7	8	194	80	0	0	26.1	0.
8	2	83	65	28	66	36.8	0.
9	2	89	90	30	0	33.5	0.

In [4]:

```
data.tail(10)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigr
1990	3	111	90	12	78	28.4	
1991	6	102	82	0	0	30.8	
1992	6	134	70	23	130	35.4	
1993	2	87	0	23	0	28.9	
1994	1	79	60	42	48	43.5	
1995	2	75	64	24	55	29.7	
1996	8	179	72	42	130	32.7	
1997	6	85	78	0	0	31.2	
1998	0	129	110	46	130	67.1	
1999	2	81	72	15	76	30.1	

In [5]:

```
data.describe()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000	2000.000000
mean	3.703500	121.182500	69.145500	20.935000	80.254000	32.193000
std	3.306063	32.068636	19.188315	16.103243	111.180534	8.149901
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	1.000000	99.000000	63.500000	0.000000	0.000000	27.375000
50%	3.000000	117.000000	72.000000	23.000000	40.000000	32.300000
75%	6.000000	141.000000	80.000000	32.000000	130.000000	36.800000
max	17.000000	199.000000	122.000000	110.000000	744.000000	80.600000

In [6]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               2000 non-null   int64  
 1   Glucose                   2000 non-null   int64  
 2   BloodPressure             2000 non-null   int64  
 3   SkinThickness             2000 non-null   int64  
 4   Insulin                   2000 non-null   int64  
 5   BMI                       2000 non-null   float64
 6   DiabetesPedigreeFunction  2000 non-null   float64
 7   Age                       2000 non-null   int64  
 8   Outcome                   2000 non-null   int64  
dtypes: float64(2), int64(7)
memory usage: 140.7 KB
```

In [7]:

```
print("no of pregencies in original dataset:" +str(len(data.index)))
```

```
no of pregencies in original dataset:2000
```

In [8]:

```
data.isnull().values.any()
```

Out[8]:

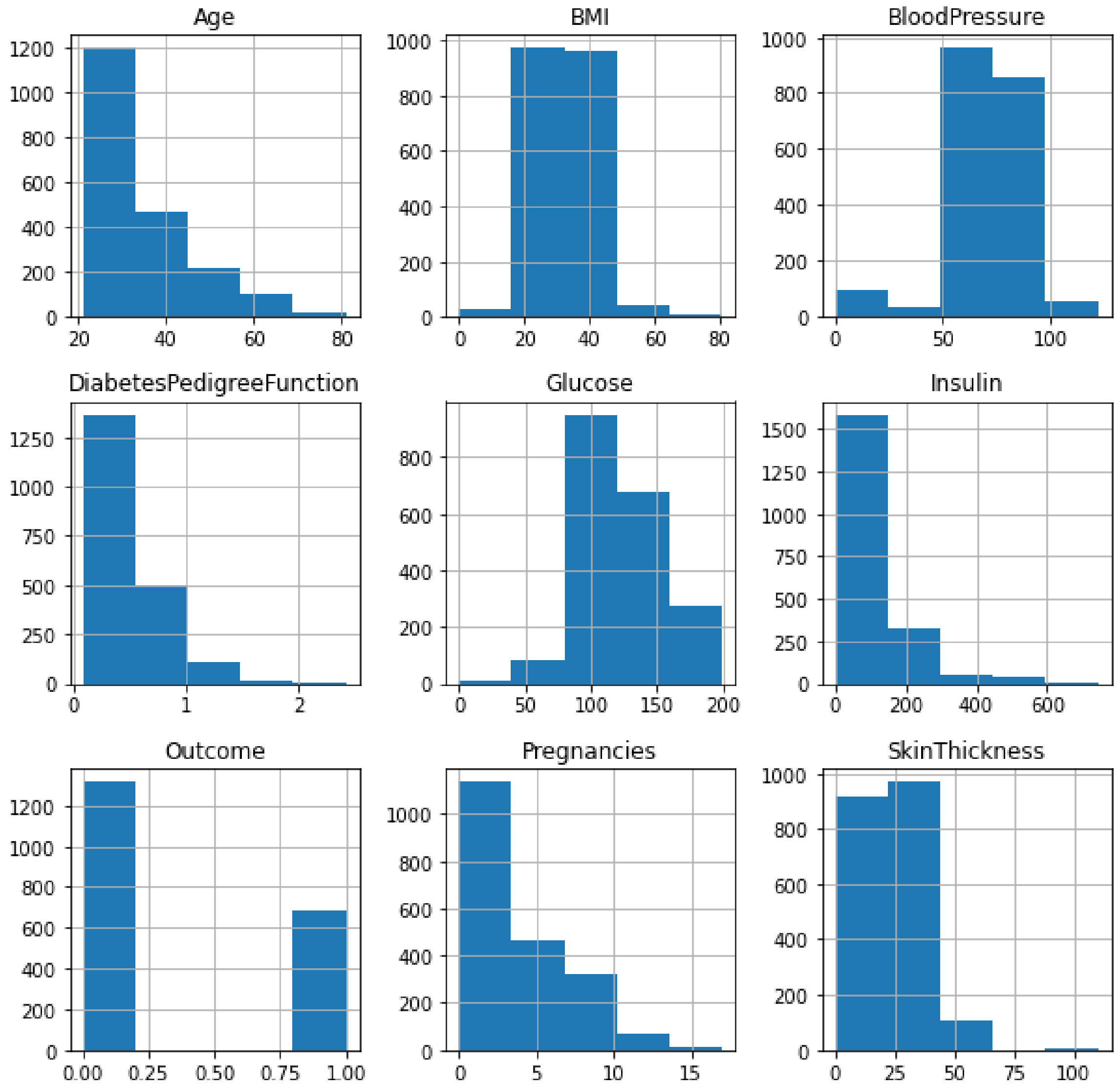
```
False
```

In [9]:

```
#histogram
data.hist(bins=5,figsize=(10,10))
```

Out[9]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x014F8478>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x01545E80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x01568898>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x01DB8298>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x01DC9C70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x01DEE610>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x01DEE688>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x01E0F0B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x06774448>]],
      dtype=object)
```



In [10]:

```
#correlation
data.corr()
```

Out[10]:

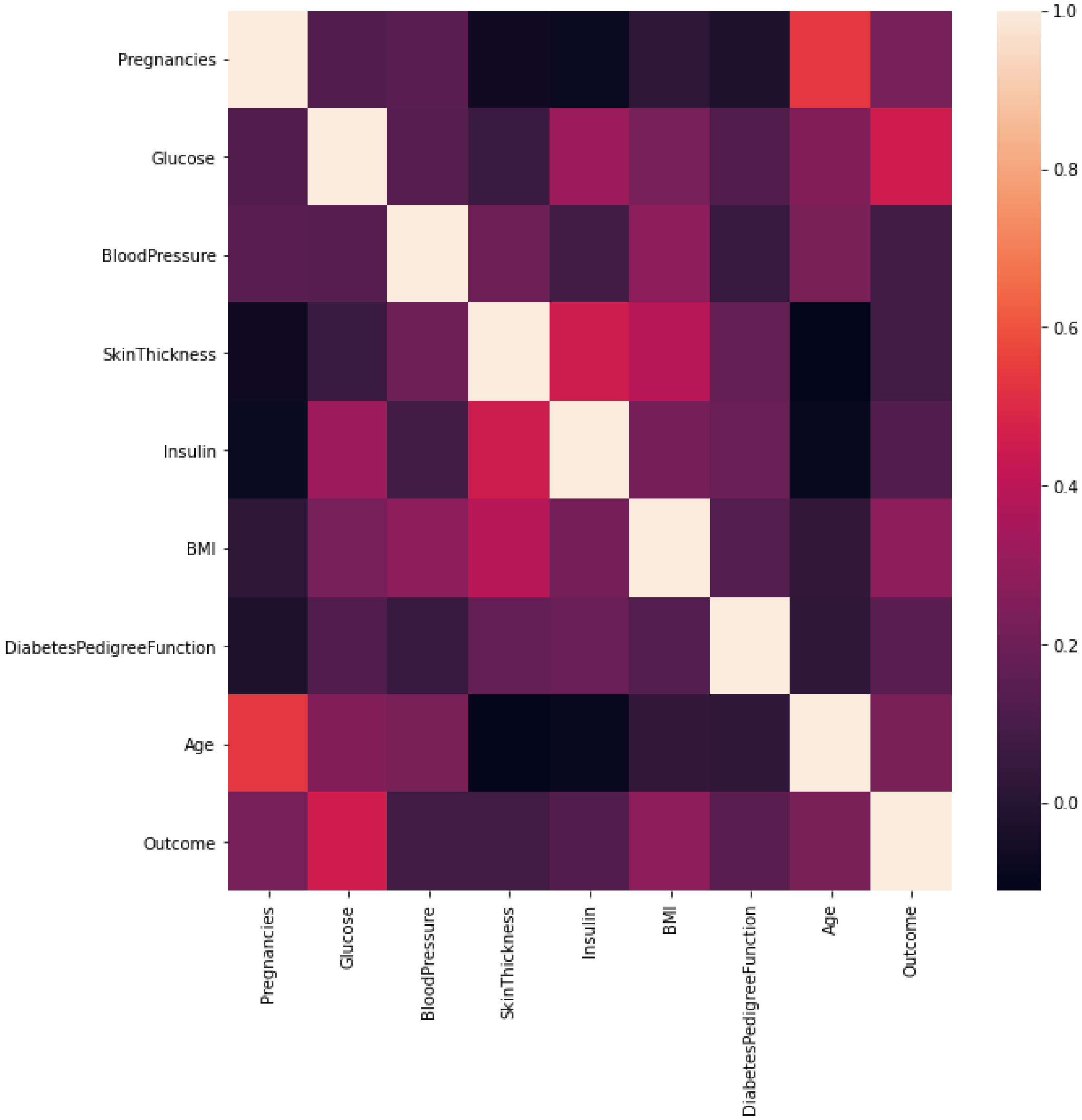
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
Pregnancies	1.000000	0.120405	0.149672	-0.063375	-0.076600
Glucose	0.120405	1.000000	0.138044	0.062368	0.320371
BloodPressure	0.149672	0.138044	1.000000	0.198800	0.087384
SkinThickness	-0.063375	0.062368	0.198800	1.000000	0.448859
Insulin	-0.076600	0.320371	0.087384	0.448859	1.000000
BMI	0.019475	0.226864	0.281545	0.393760	0.223012
DiabetesPedigreeFunction	-0.025453	0.123243	0.051331	0.178299	0.192719
Age	0.539457	0.254496	0.238375	-0.111034	-0.085879
Outcome	0.224437	0.458421	0.075958	0.076040	0.120924

In [11]:

```
#correlation using heat maps
plt.figure(figsize=(10,10))
sns.heatmap(data.corr())
# we can see skin thickness,insulin,pregnencies and age are full independent to each ot
her
#age and pregencies has negative correlation
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xa59ad00>
```

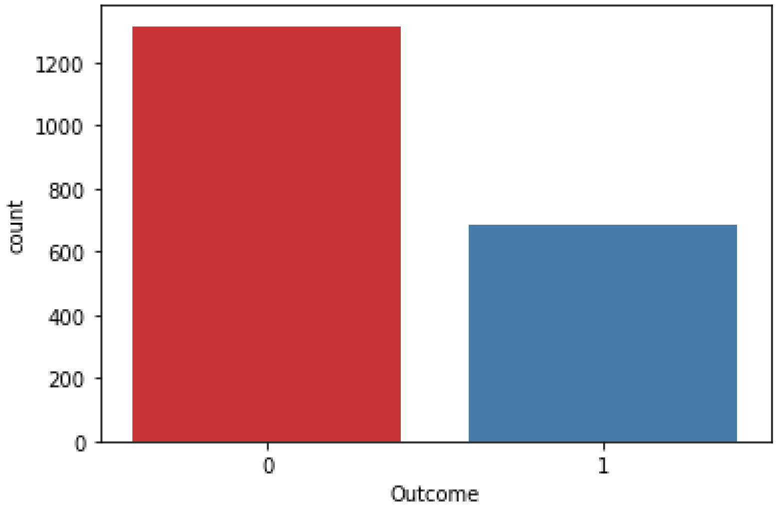

Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome
1.0
0.8
0.6
0.4
0.2
0.0

In [12]:

```
sns.countplot(x=data['Outcome'],palette='Set1')
#0 means no diabeted
#1 means patient with diabtes
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb043d90>
```

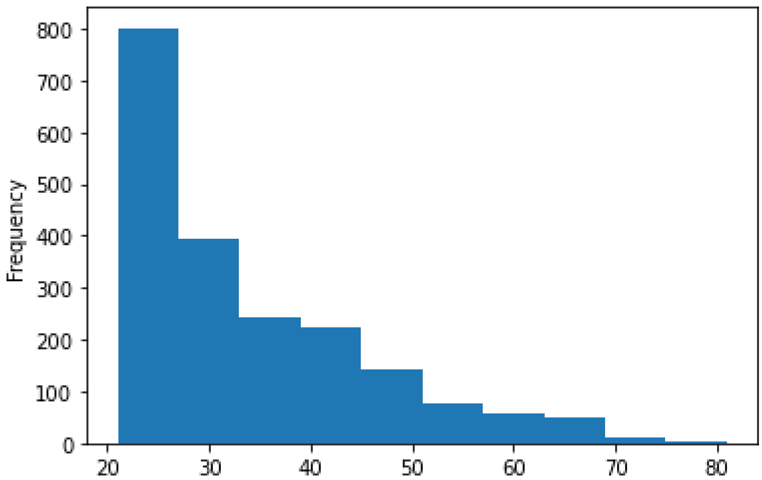


In [13]:

```
data["Age"].plot.hist()
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0xb0786e8>
```



In [14]:

```
data.isnull().sum()
```

Out[14]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [15]:

```
sns.set(style="ticks")
sns.pairplot(data, hue="Outcome")
```

Out[15]:

```
<seaborn.axisgrid.PairGrid at 0x688f8b0>
```

In [16]:

```
#outlier remove
Q1=data.quantile(0.25)
Q3=data.quantile(0.75)
IQR=Q3-Q1

print("****Q1**** \n",Q1)
print("\n****Q3**** \n",Q3)
print("\n****IQR****\n",IQR)

#print((df < (Q1 - 1.5 * IQR))|(df > (Q3 + 1.5 * IQR)))
```

```
****Q1****
 Pregnancies                  1.000
Glucose                     99.000
BloodPressure               63.500
SkinThickness                0.000
Insulin                      0.000
BMI                         27.375
DiabetesPedigreeFunction     0.244
Age                         24.000
Outcome                      0.000
Name: 0.25, dtype: float64

****Q3****
 Pregnancies                  6.000
Glucose                     141.000
BloodPressure                80.000
SkinThickness                32.000
Insulin                     130.000
BMI                          36.800
DiabetesPedigreeFunction      0.624
Age                          40.000
Outcome                       1.000
Name: 0.75, dtype: float64

****IQR****
 Pregnancies                  5.000
Glucose                      42.000
BloodPressure                16.500
SkinThickness                32.000
Insulin                     130.000
BMI                           9.425
DiabetesPedigreeFunction      0.380
Age                          16.000
Outcome                       1.000
dtype: float64
```

In [17]:

```
data_out = data[~((data < (Q1 - 1.5 * IQR)) |(data> (Q3 + 1.5 * IQR))).any(axis=1)]
data.shape,data_out.shape
```

Out[17]:

```
((2000, 9), (1652, 9))
```

In [19]:

```
#Scatter matrix after removing outlier
sns.set(style="ticks")
sns.pairplot(data_out, hue="Outcome")
plt.show()
```

In [20]:

```
#lets extract features and targets
X=data_out.drop(columns=['Outcome'])
y=data_out['Outcome']
```

In [21]:

```
#Splitting train test data 80 20 ratio
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,test_y=train_test_split(X,y,test_size=0.2)
```

In [22]:

```
len(train_X)
```

Out[22]:

1321

In [23]:

```
len(train_y)
```

Out[23]:

```
1321
```

In [24]:

```
len(test_X)
```

Out[24]:

```
331
```

In [25]:

```
len(test_y)
```

Out[25]:

```
331
```

In [26]:

```
train_X.shape,test_X.shape,train_y.shape,test_y.shape
```

Out[26]:

```
((1321, 8), (331, 8), (1321,), (331,))
```

In [27]:

```
from sklearn.metrics import confusion_matrix,accuracy_score,make_scorer
from sklearn.model_selection import cross_validate

def tn(y_true, y_pred): return confusion_matrix(y_true, y_pred)[0, 0]
def fp(y_true, y_pred): return confusion_matrix(y_true, y_pred)[0, 1]
def fn(y_true, y_pred): return confusion_matrix(y_true, y_pred)[1, 0]
def tp(y_true, y_pred): return confusion_matrix(y_true, y_pred)[1, 1]

#cross validation purpose
scoring = {'accuracy': make_scorer(accuracy_score),'prec': 'precision'}
scoring = {'tp': make_scorer(tp), 'tn': make_scorer(tn),
           'fp': make_scorer(fp), 'fn': make_scorer(fn)}

def display_result(result):
    print("TP: ",result['test_tp'])
    print("TN: ",result['test_tn'])
    print("FN: ",result['test_fn'])
    print("FP: ",result['test_fp'])
```

In [28]:

```
#Lets build the model
#Logistic Regression
#s(z)=1/(1+e^-z) z=input
import warnings
warnings.filterwarnings("ignore")
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score
acc=[]
roc=[]
clf=LogisticRegression()
clf.fit(train_X,train_y)
y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
acc.append(ac)

#find the ROC_AOC curve
rc=roc_auc_score(test_y,y_pred)
roc.append(rc)
print("\nAccuracy {0} ROC {1}".format(ac,rc))
#cross val score
result=cross_validate(clf,train_X,train_y,scoring=scoring,cv=10)
display_result(result)

#display predicted values uncomment below line
pd.DataFrame(data={'Actual':test_y,'Predicted':y_pred}).head()
```

```
Accuracy 0.797583081570997 ROC 0.7342032967032968
TP:  [25 19 22 26 24 26 21 25 25 23]
TN:  [80 83 82 78 77 85 78 85 83 80]
FN:  [18 24 21 17 19 16 21 17 17 19]
FP:  [10  6  7 11 12  5 12  5  7 10]
```

Out[28]:

	Actual	Predicted
62	0	0
1850	0	0
851	0	0
969	0	1
1280	0	0

In [29]:

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
clf=KNeighborsClassifier(n_neighbors=3)
clf.fit(train_X,train_y)
y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
acc.append(ac)

#find the ROC_AOC curve
rc=roc_auc_score(test_y,y_pred)
roc.append(rc)
print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score
result=cross_validate(clf,train_X,train_y,scoring=scoring,cv=10)
display_result(result)

#display predicted values uncomment below line
pd.DataFrame(data={'Actual':test_y,'Predicted':y_pred}).head()
```

```
Accuracy 0.8670694864048338 ROC 0.8435210622710622
TP:  [38 32 36 31 32 33 31 33 35 34]
TN:  [75 77 82 83 79 85 77 81 81 77]
FN:  [ 5 11  7 12 11  9 11  9  7  8]
FP:  [15 12  7  6 10  5 13  9  9 13]
```

Out[29]:

	Actual	Predicted
62	0	0
1850	0	0
851	0	0
969	0	0
1280	0	0

In [30]:

```
#Random forest
from sklearn.ensemble import RandomForestClassifier

clf=RandomForestClassifier()
clf.fit(train_X,train_y)

y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
acc.append(ac)

#find the ROC_AOC curve
rc=roc_auc_score(test_y,y_pred)
roc.append(rc)
print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score
result=cross_validate(clf,train_X,train_y,scoring=scoring,cv=10)
display_result(result)

#display predicted values uncomment below line
pd.DataFrame(data={'Actual':test_y,'Predicted':y_pred}).head()
```

```
Accuracy 0.9758308157099698 ROC 0.9696886446886447
TP:  [41 40 41 39 40 41 41 41 41 39]
TN:  [85 88 86 87 87 89 89 89 87 88]
FN:  [2 3 2 4 3 1 1 1 1 3]
FP:  [5 1 3 2 2 1 1 1 3 2]
```

Out[30]:

	Actual	Predicted
62	0	0
1850	0	0
851	0	0
969	0	0
1280	0	0

In [31]:

```
#Naive Bayes Theorem
#import library
from sklearn.naive_bayes import GaussianNB

clf=GaussianNB()
clf.fit(train_X,train_y)
y_pred=clf.predict(test_X)
#find accuracy
ac=accuracy_score(test_y,y_pred)
acc.append(ac)

#find the ROC_AOC curve
rc=roc_auc_score(test_y,y_pred)
roc.append(rc)
print("\nAccuracy {0} ROC {1}".format(ac,rc))

#cross val score
result=cross_validate(clf,train_X,train_y,scoring=scoring,cv=10)
display_result(result)

#display predicted values uncomment below line
pd.DataFrame(data={'Actual':test_y,'Predicted':y_pred}).head()
```

```
Accuracy 0.7492447129909365 ROC 0.7179258241758241
TP:  [27 22 27 29 24 28 24 29 27 25]
TN:  [79 77 75 71 72 74 80 73 79 75]
FN:  [16 21 16 14 19 14 18 13 15 17]
FP:  [11 12 14 18 17 16 10 17 11 15]
```

Out[31]:

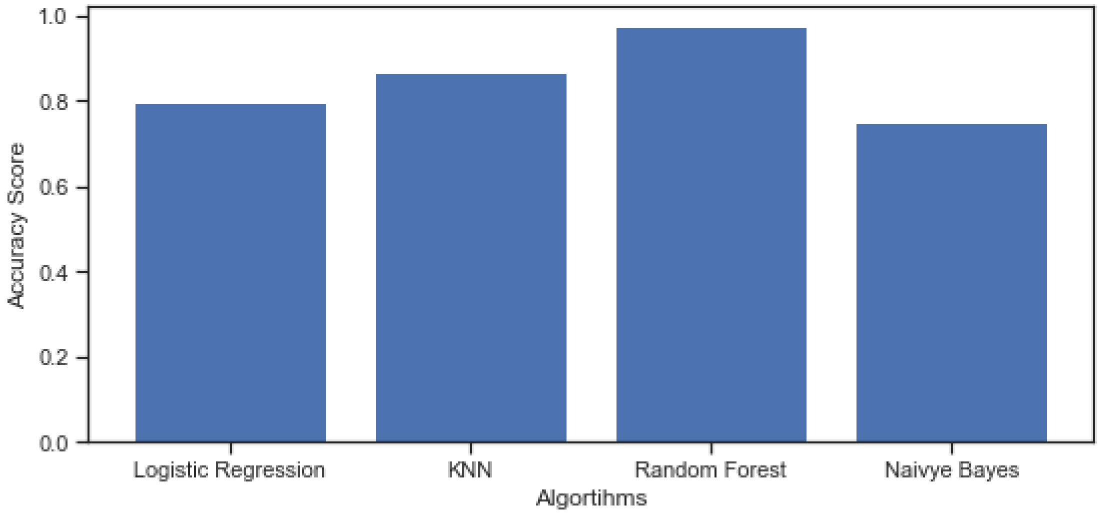
	Actual	Predicted
62	0	0
1850	0	0
851	0	0
969	0	1
1280	0	0

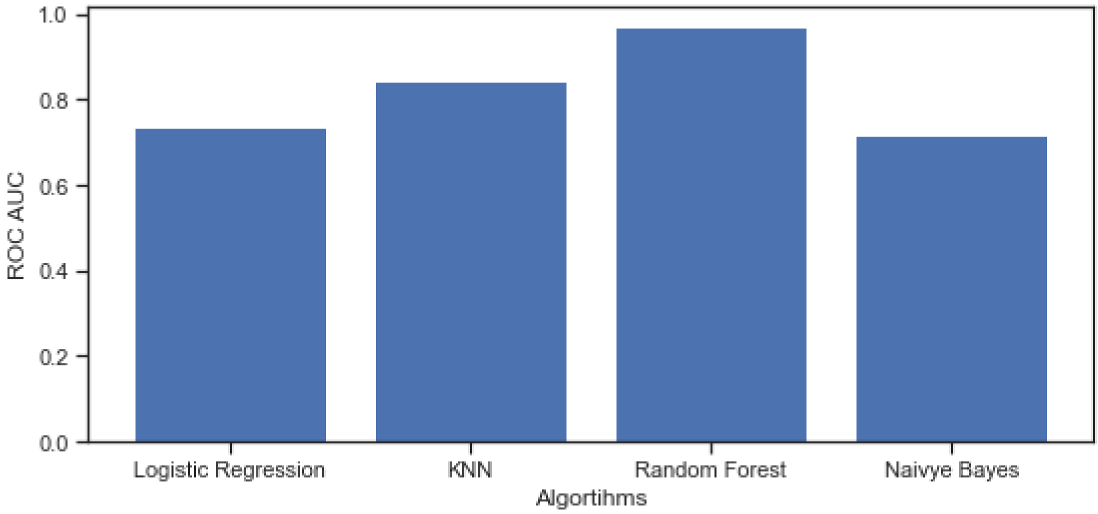
In [32]:

```
#lets plot the bar graph

ax=plt.figure(figsize=(9,4))
plt.bar(['Logistic Regression','KNN','Random Forest','Naivye Bayes'],acc,label='Accurac
y')
plt.ylabel('Accuracy Score')
plt.xlabel('Algortihms')
plt.show()

ax=plt.figure(figsize=(9,4))
plt.bar(['Logistic Regression','KNN','Random Forest','Naivye Bayes'],roc,label='ROC AU
C')
plt.ylabel('ROC AUC')
plt.xlabel('Algortihms')
plt.show()
```





In [ ]:

```
#Random forest has highest accuracy 98% and ROC_AUC curve 97%
#model can be improve more if we take same count of labels
#in our model 30% is diabetic and 70% no diabetic patient

#model can be improve with fine tunning
```