In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
df=pd.read_csv('diabetes.csv')
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Ag
0	2	138	62	35	0	33.6	0.127	
1	0	84	82	31	125	38.2	0.233	
2	0	145	0	0	0	44.2	0.630	
3	0	135	68	42	250	42.3	0.365	
4	1	139	62	41	480	40.7	0.536	

In [3]:
```
#lets describe the data
df.describe()
```

Out[3]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
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	

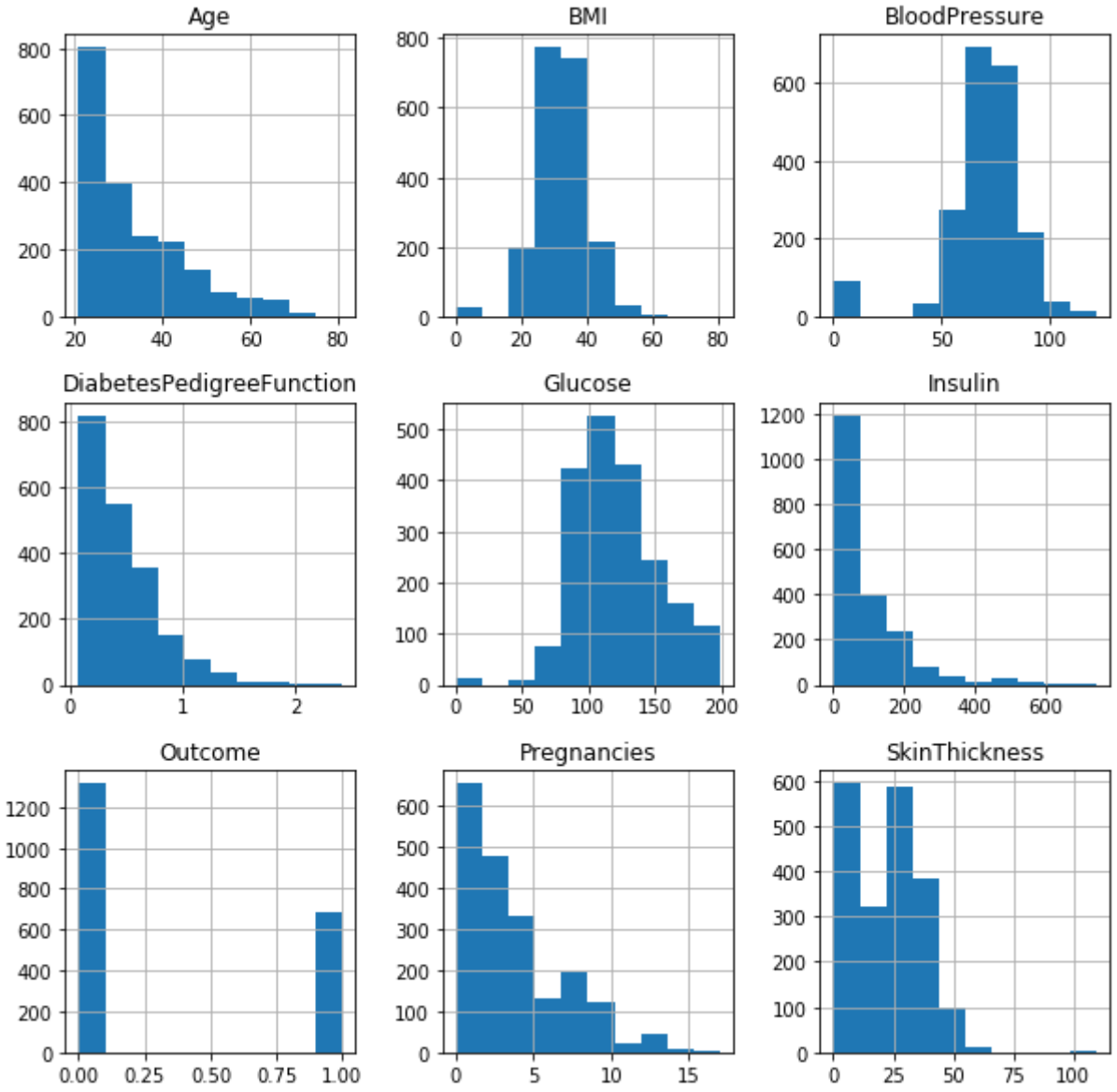
In [4]:
```
#infromation of dataset
df.info()
```

```
<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: 140.8 KB
```

In [5]:
```
#any null values
#not neccessary in above information we can see
df.isnull().values.any()
```

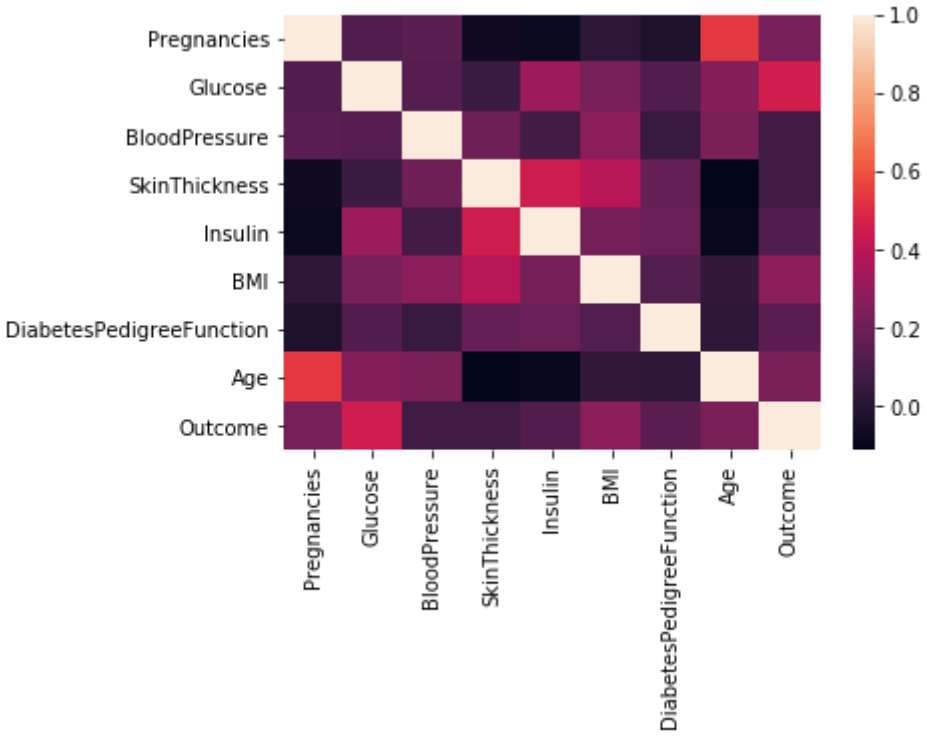
Out[5]:
```
False
```

In [6]:
```
#histogram
df.hist(bins=10,figsize=(10,10))
plt.show()
```



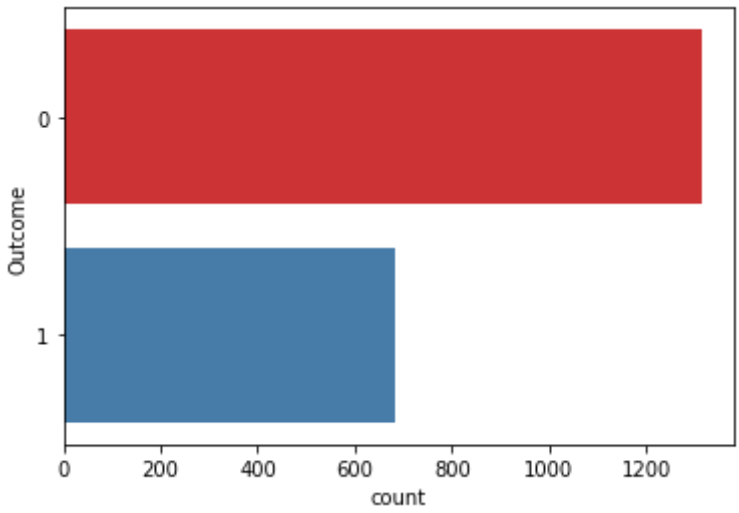
In [7]:
```
#correlation

sns.heatmap(df.corr())
# we can see skin thickness,insulin,pregnencies and age are full indepe
#age and pregencies has negative correlation
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd261b32e50>



In [8]:
```
#lets count total outcome in each target 0 1
#0 means no diabeted
#1 means patient with diabtes
sns.countplot(y=df['Outcome'],palette='Set1')
```

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd261bd7390>

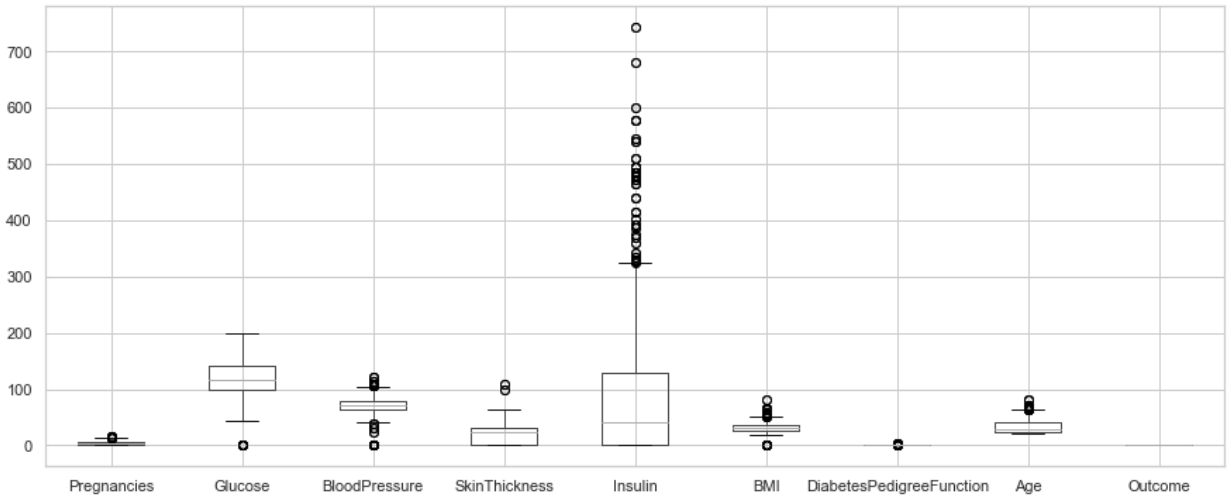


In [9]:
```
1 sns.set(style="ticks")
2 sns.pairplot(df, hue="Outcome")
```

Out[9]: <seaborn.axisgrid.PairGrid at 0x7fd261c58c10>

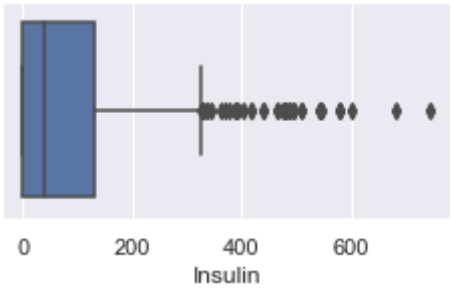
In [10]:
```
#box plot for outlier visualization
sns.set(style="whitegrid")
df.boxplot(figsize=(15,6))
```

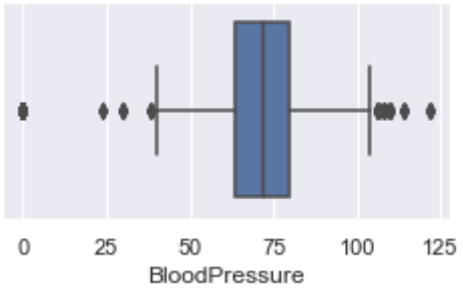
Out[10]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd2641fcf50>

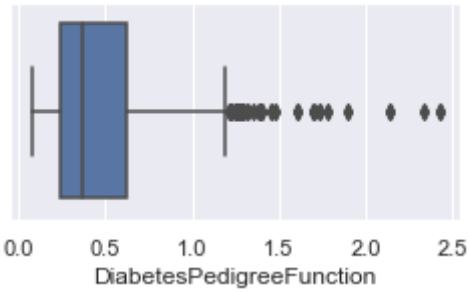


In [11]:

```
#box plot
sns.set(style="whitegrid")

sns.set(rc={'figure.figsize':(4,2)})
sns.boxplot(x=df['Insulin'])
plt.show()
sns.boxplot(x=df['BloodPressure'])
plt.show()
sns.boxplot(x=df['DiabetesPedigreeFunction'])
plt.show()
```







In [12]:
```
#outlier remove

Q1=df.quantile(0.25)
Q3=df.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 [13]:
```
#outlier remove
df_out = df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axi
df.shape,df_out.shape
#more than 80 records deleted
```

Out[13]:
```
((2000, 9), (1652, 9))
```

In [14]:
```
#Scatter matrix after removing outlier
sns.set(style="ticks")
sns.pairplot(df_out, hue="Outcome")
plt.show()
```

In [15]:
```
#lets extract features and targets
X=df_out.drop(columns=['Outcome'])
y=df_out['Outcome']
```

In [16]:
```
#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 [17]:
```
train_X.shape,test_X.shape,train_y.shape,test_y.shape
```

Out[17]: ((1321, 8), (331, 8), (1321,), (331,))

In [18]:

```
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 [19]:
```
#Lets build the model

#Logistic Regression
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()
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/roshanremanan/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/_logistic.py:940: 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://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/Users/roshanremanan/opt/anaconda3/lib/python3.7/site-packages/sklearn/li
```

In [20]:

```
#Support Vector Machine
from sklearn.svm import SVC

clf=SVC(kernel='linear')
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 0.8187311178247734 ROC 0.7394548063127689
TP:  [26 25 21 22 18 20 20 23 24 26]
TN:  [86 80 80 79 77 76 81 81 81 87]
FN:  [18 18 22 21 25 23 23 20 19 17]
FP:  [ 3  9  9 10 12 13  8  8  8  2]
```

In [21]:

```
#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 0.8972809667673716 ROC 0.9039454806312768
TP:  [34 33 34 37 29 29 30 29 30 36]
TN:  [87 81 79 76 79 80 78 81 85 82]
FN:  [10 10  9  6 14 14 13 14 13  7]
FP:  [ 2  8 10 13 10  9 11  8  4  7]
```

In [22]:

```
#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 0.972809667673716 ROC 0.9817073170731707
TP:  [43 42 42 40 42 41 43 41 38 41]
TN:  [88 89 86 84 85 88 89 87 89 87]
FN:  [1 1 1 3 1 2 0 2 5 2]
FP:  [1 0 3 5 4 1 0 2 0 2]
```

In [23]:

```
#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()
```

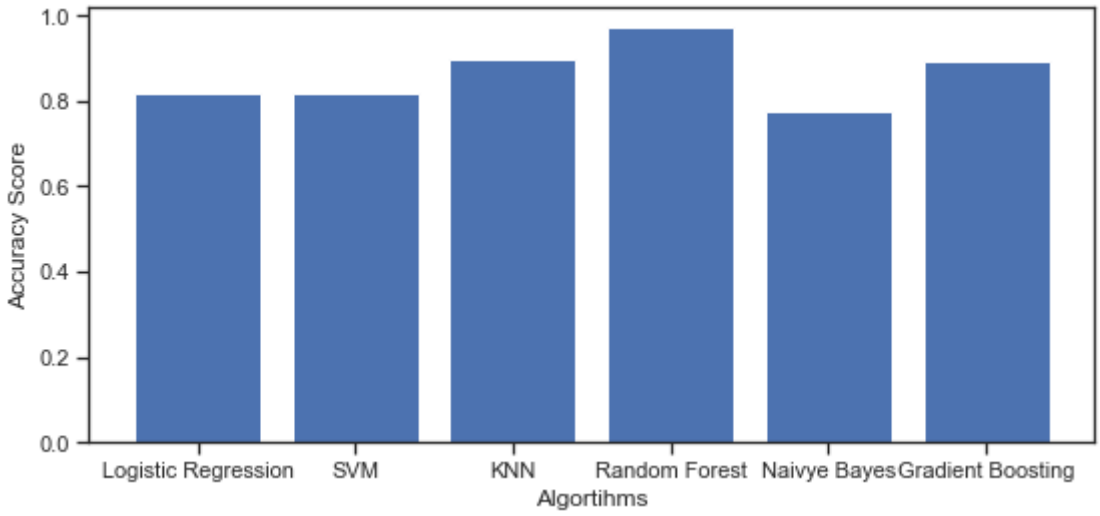
```
Accuracy 0.7764350453172205 ROC 0.7379483500717361
TP:  [25 28 29 24 23 25 25 28 27 27]
TN:  [78 73 75 76 71 75 76 77 76 79]
FN:  [19 15 14 19 20 18 18 15 16 16]
FP:  [11 16 14 13 18 14 13 12 13 10]
```

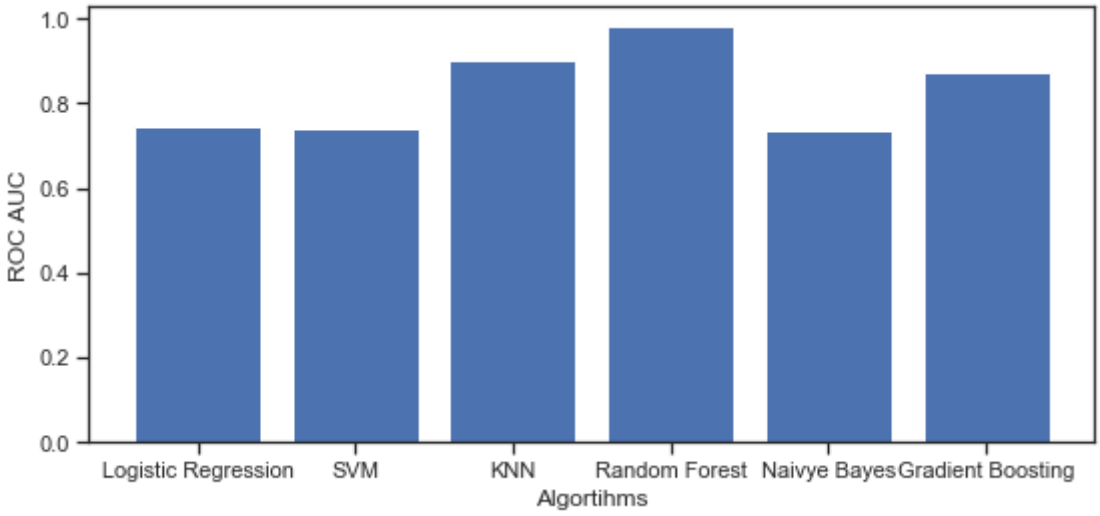
In [24]:

```
#Gradient Boosting Classifier
from sklearn.ensemble import GradientBoostingClassifier
clf=GradientBoostingClassifier(n_estimators=50,learning_rate=0.2)
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 0.8912386706948641 ROC 0.8729316116690579
TP:  [35 36 30 33 37 29 31 33 36 33]
TN:  [86 89 83 81 83 83 81 87 84 82]
FN:  [ 9  7 13 10  6 14 12 10  7 10]
FP:  [3 0 6 8 6 6 8 2 5 7]
```

In [25]:
```
#lets plot the bar graph

ax=plt.figure(figsize=(9,4))
plt.bar(['Logistic Regression','SVM','KNN','Random Forest','Naivye Baye
plt.ylabel('Accuracy Score')
plt.xlabel('Algortihms')
plt.show()

ax=plt.figure(figsize=(9,4))
plt.bar(['Logistic Regression','SVM','KNN','Random Forest','Naivye Baye
plt.ylabel('ROC AUC')
plt.xlabel('Algortihms')
plt.show()
```





In [26]:
```
#Great....
#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
```

In [ ]:
```

```

In [ ]:
```
1
```