--------||TASK 2: DIABETES PREDICTION||-------

*Importing the libraries and seeking the dataset

In [64]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.simplefilter(action = "ignore")
diabetes = pd.read_csv('diabetes.csv')
print(diabetes.columns)
```

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [65]:
```
diabetes.head()
```

Out[65]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	O
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	
3	1	89	66	23	94	28.1	0.167	21	
4	0	137	40	35	168	43.1	2.288	33	

In [66]:
```
print("dimension of diabetes data: {}".format(diabetes.shape))
```

```
dimension of diabetes data: (768, 9)
```

In [67]:
```
print(diabetes.groupby('Outcome').size())
```

```
Outcome
0    500
1    268
dtype: int64
```

In [68]:
```
diabetes.info()
```

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

```
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [69]:

```
# Now, we can look at where are missing values
diabetes.isnull().sum()
```

Out[69]:

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

In [101…

```
# In the data set, there were asked whether there were any outlier observations compare
# It was found to be an outlier observation.
for feature in diabetes:

    Q1 = diabetes[feature].quantile(0.25)
    Q3 = diabetes[feature].quantile(0.75)
    IQR = Q3-Q1
    lower = Q1- 1.5*IQR
    upper = Q3 + 1.5*IQR

    if diabetes[(diabetes[feature] > upper)].any(axis=None):
        print(feature,"yes")
    else:
        print(feature, "no")
```

```
Pregnancies yes
Glucose no
BloodPressure yes
SkinThickness yes
Insulin yes
BMI yes
DiabetesPedigreeFunction yes
Age yes
Outcome no
```

In [71]:

```
# It consists of 768 observation units and 9 variables.
diabetes.shape
```

Out[71]:

```
(768, 9)
```

In [72]:
```
# Checking for null data if present
diabetes.isnull().values.any()
```

Out[72]:
```
False
```

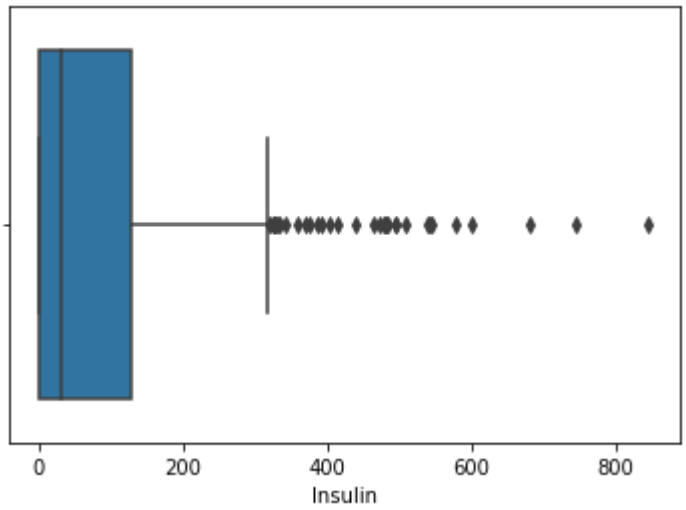
In [73]:
```
# Displaying the data types
diabetes.dtypes
```

Out[73]:
```
Pregnancies                 int64
Glucose                     int64
BloodPressure               int64
SkinThickness               int64
Insulin                     int64
BMI                       float64
DiabetesPedigreeFunction  float64
Age                         int64
Outcome                     int64
dtype: object
```

*Displaying graphs and descrbing the dataset

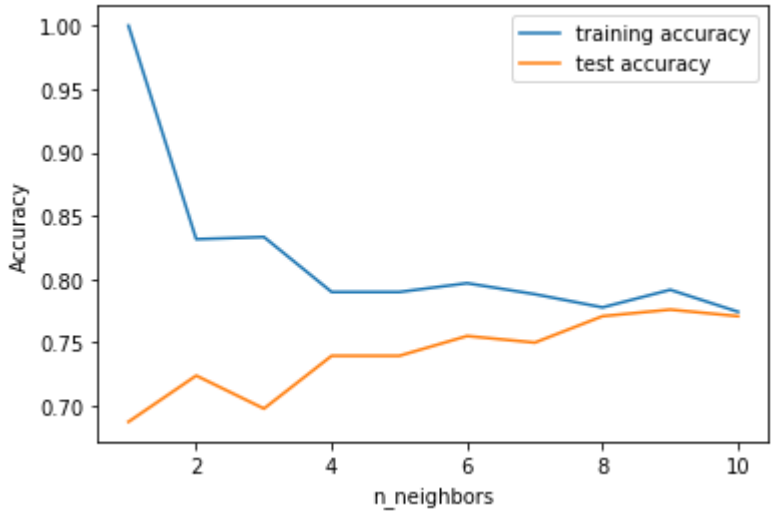
In [74]:
```
sns.boxplot(x = diabetes["Insulin"])
```

Out[74]:
```
<AxesSubplot:xlabel='Insulin'>
```



In [75]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(diabetes.loc[:, diabetes.columns !=
from sklearn.neighbors import KNeighborsClassifier
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # build the model
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # record test set accuracy
    test_accuracy.append(knn.score(X_test, y_test))
```

```
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```



In [76]:
```
# Descriptive statistics of the data set accessed.
diabetes.describe([0.10,0.25,0.50,0.75,0.90,0.95,0.99]).T
```

Out[76]:

	count	mean	std	min	10%	25%	50%	75%
Pregnancies	768.0	3.845052	3.369578	0.000	0.000	1.00000	3.0000	6.00000
Glucose	768.0	120.894531	31.972618	0.000	85.000	99.00000	117.0000	140.25000
BloodPressure	768.0	69.105469	19.355807	0.000	54.000	62.00000	72.0000	80.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.000	0.00000	23.0000	32.00000
Insulin	768.0	79.799479	115.244002	0.000	0.000	0.00000	30.5000	127.25000
BMI	768.0	31.992578	7.884160	0.000	23.600	27.30000	32.0000	36.60000
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.165	0.24375	0.3725	0.62625
Age	768.0	33.240885	11.760232	21.000	22.000	24.00000	29.0000	41.00000
Outcome	768.0	0.348958	0.476951	0.000	0.000	0.00000	0.0000	1.00000

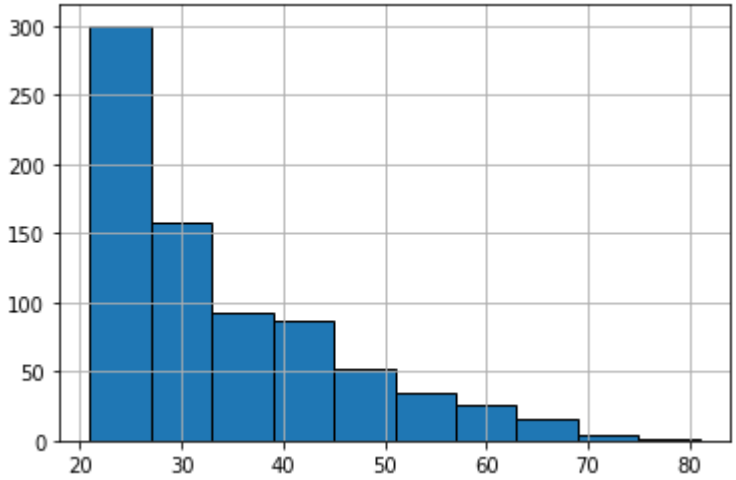
In [77]:
```
# The distribution of the Outcome variable was examined.
diabetes["Outcome"].value_counts()*100/len(diabetes)
```

Out[77]:
```
0    65.104167
1    34.895833
Name: Outcome, dtype: float64
```

In [78]:
```
# The classes of the outcome variable were examined.
diabetes.Outcome.value_counts()
```

Out[78]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [79]:
```
# The histagram of the Age variable was reached.
diabetes["Age"].hist(edgecolor = "black");
```

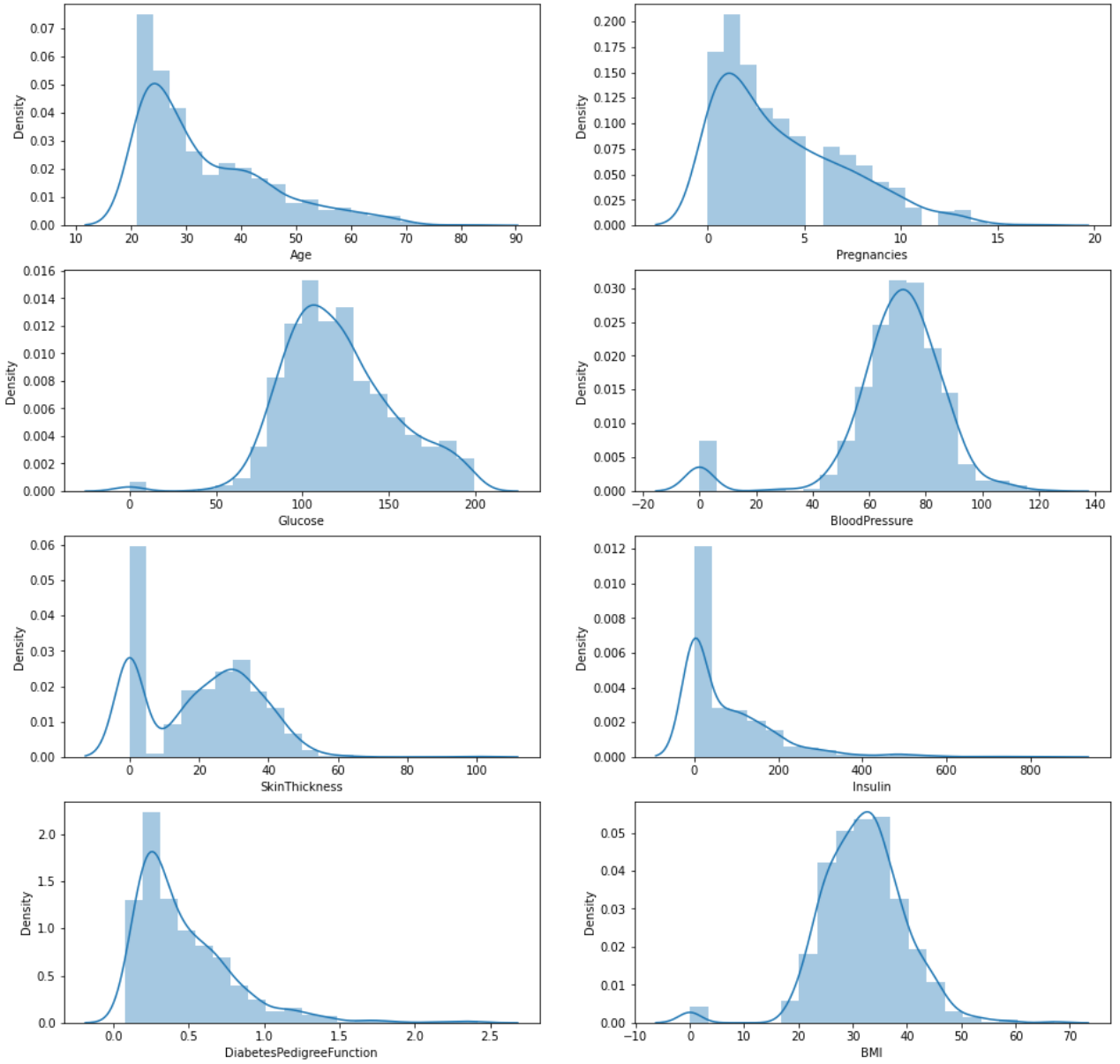


In [80]:
```
print("Max Age: " + str(diabetes["Age"].max()) + " Min Age: " + str(diabetes["Age"].min
```

```
Max Age: 81 Min Age: 21
```

In [81]:
```
# Histogram and density graphs of all variables were accessed.
fig, ax = plt.subplots(4,2, figsize=(16,16))
sns.distplot(diabetes.Age, bins = 20, ax=ax[0,0])
sns.distplot(diabetes.Pregnancies, bins = 20, ax=ax[0,1])
sns.distplot(diabetes.Glucose, bins = 20, ax=ax[1,0])
sns.distplot(diabetes.BloodPressure, bins = 20, ax=ax[1,1])
sns.distplot(diabetes.SkinThickness, bins = 20, ax=ax[2,0])
sns.distplot(diabetes.Insulin, bins = 20, ax=ax[2,1])
sns.distplot(diabetes.DiabetesPedigreeFunction, bins = 20, ax=ax[3,0])
sns.distplot(diabetes.BMI, bins = 20, ax=ax[3,1])
```

Out[81]:
```
<AxesSubplot:xlabel='BMI', ylabel='Density'>
```



In [82]:
```
diabetes.groupby("Outcome").agg({"Pregnancies":"mean"})
```

Out[82]:

Outcome	Pregnancies
0	3.298000
1	4.865672

In [83]:
```
diabetes.groupby("Outcome").agg({"Age":"mean"})
```

Out[83]:

Outcome	Age
0	31.190000
1	37.067164

In [84]:
```
diabetes.groupby("Outcome").agg({"Age":"max"})
```

Out[84]:

Outcome	Age
0	81
1	70

In [85]:
```
diabetes.groupby("Outcome").agg({"Insulin": "mean"})
```

Out[85]:

Outcome	Insulin
0	68.792000
1	100.335821

In [86]:
```
diabetes.groupby("Outcome").agg({"Insulin": "max"})
```

Out[86]:

Outcome	Insulin
0	744
1	846

In [87]:
```
diabetes.groupby("Outcome").agg({"Glucose": "mean"})
```

Out[87]:

Outcome	Glucose
0	109.980000
1	141.257463

In [88]:
```
diabetes.groupby("Outcome").agg({"Glucose": "max"})
```

Out[88]:

Outcome	Glucose
0	197
1	199

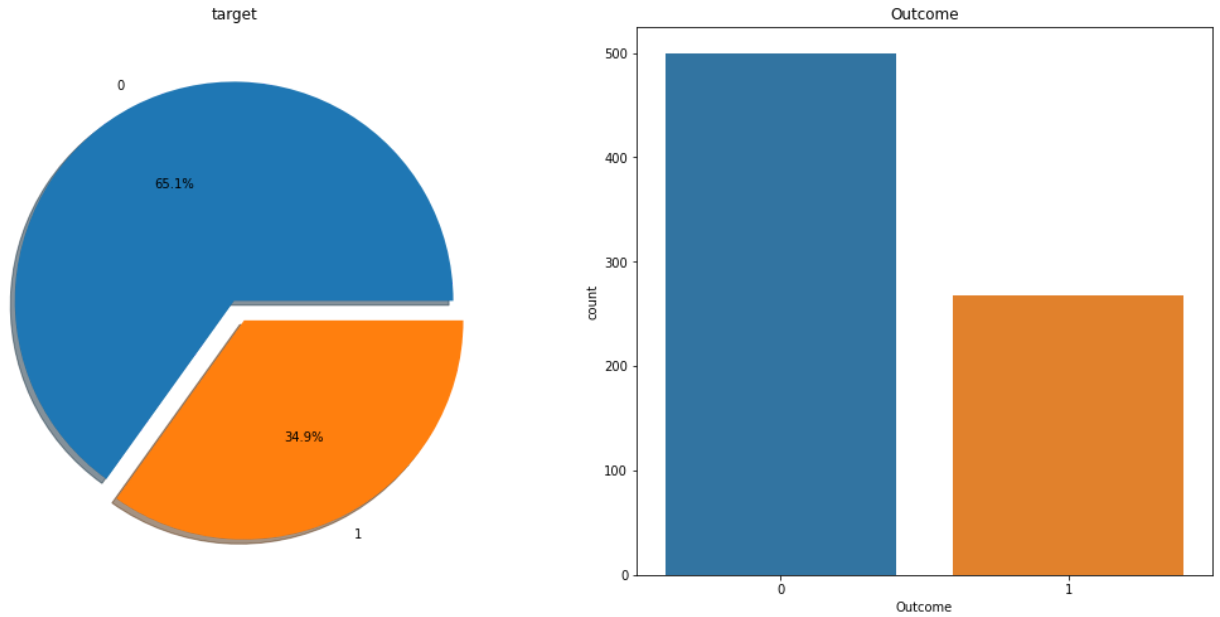
In [89]:
```
diabetes.groupby("Outcome").agg({"BMI": "mean"})
```

Out[89]:

Outcome	BMI
0	30.304200
1	35.142537

In [90]:

```
# The distribution of the outcome variable in the data was examined and visualized.
f,ax=plt.subplots(1,2,figsize=(18,8))
diabetes['Outcome'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',ax=ax[0],
ax[0].set_title('target')
ax[0].set_ylabel('')
sns.countplot('Outcome',data=diabetes,ax=ax[1])
ax[1].set_title('Outcome')
plt.show()
```



In [91]:

```
# Access to the correlation of the data set was provided. What kind of relationship is
# If the correlation value is> 0, there is a positive correlation. While the value of o
# Correlation = 0 means no correlation.
# If the correlation is <0, there is a negative correlation. While one variable increas
# When the correlations are examined, there are 2 variables that act as a positive corr
# These variables are Glucose. As these increase, Outcome variable increases.
diabetes.corr()
```

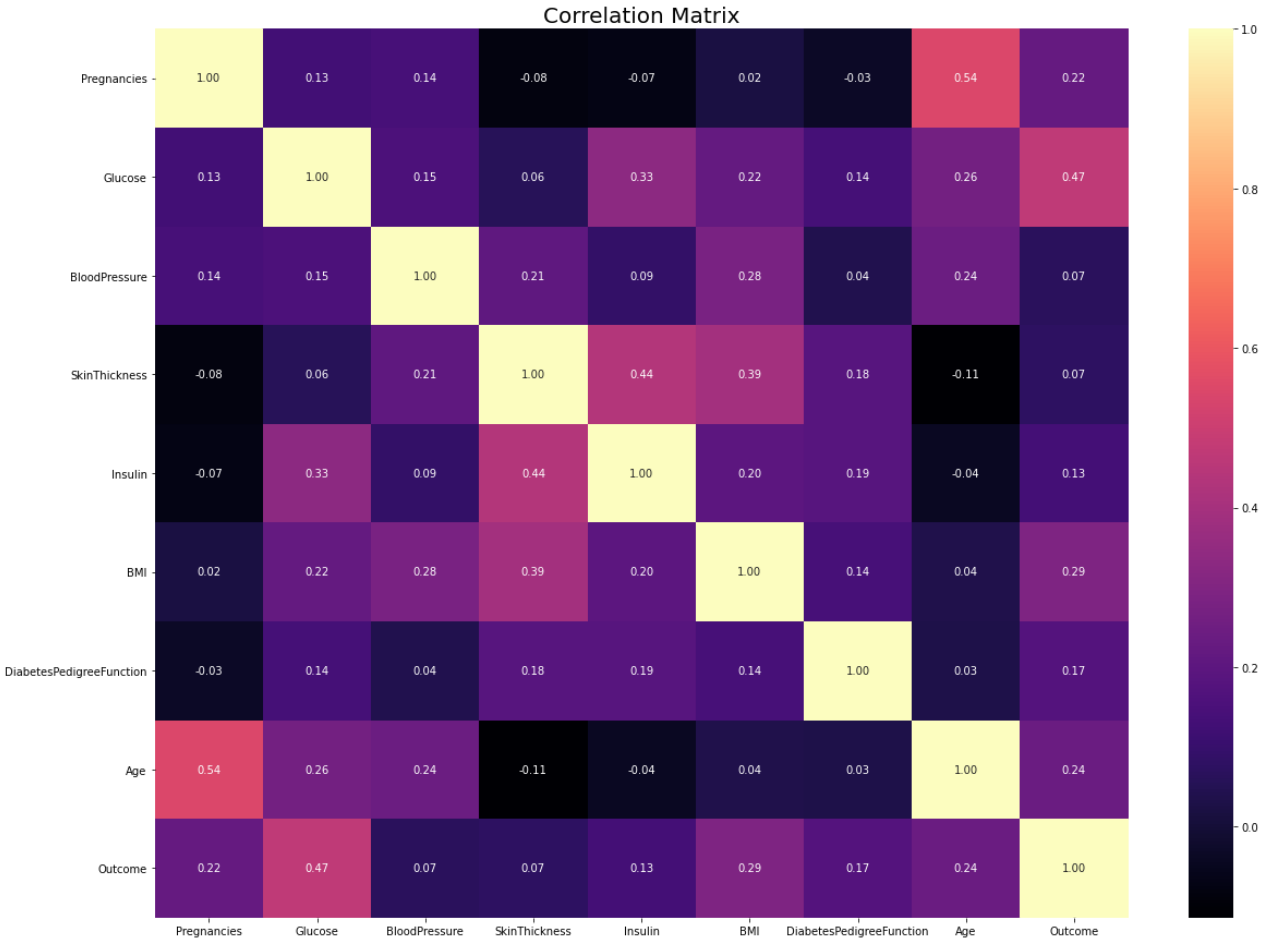
Out[91]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	

In [92]:

```
# Correlation matrix graph of the data set
f, ax = plt.subplots(figsize= [20,15])
sns.heatmap(diabetes.corr(), annot=True, fmt=".2f", ax=ax, cmap = "magma" )
ax.set_title("Correlation Matrix", fontsize=20)
plt.show()
```



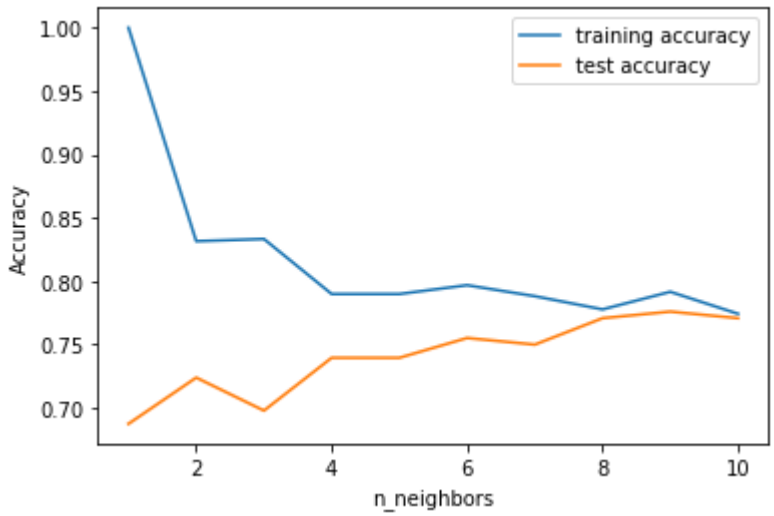
*Machine Learning Algorithms

1. k-Nearest Neighbours

In [93]:

```
X_train, X_test, y_train, y_test = train_test_split(diabetes.loc[:, diabetes.columns !=
training_accuracy = []
test_accuracy = []
# try n_neighbors from 1 to 10
neighbors_settings = range(1, 11)
for n_neighbors in neighbors_settings:
    # build the model
```

```
    knn = KNeighborsClassifier(n_neighbors=n_neighbors)
    knn.fit(X_train, y_train)
    # record training set accuracy
    training_accuracy.append(knn.score(X_train, y_train))
    # record test set accuracy
    test_accuracy.append(knn.score(X_test, y_test))
plt.plot(neighbors_settings, training_accuracy, label="training accuracy")
plt.plot(neighbors_settings, test_accuracy, label="test accuracy")
plt.ylabel("Accuracy")
plt.xlabel("n_neighbors")
plt.legend()
plt.savefig('knn_compare_model')
```



In [94]:

```
knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(X_train, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test
```

```
Accuracy of K-NN classifier on training set: 0.79
Accuracy of K-NN classifier on test set: 0.78
```

2. Logistic Regression

In [95]:

```
logreg = LogisticRegression().fit(X_train, y_train)
print("Training set score: {:.3f}".format(logreg.score(X_train, y_train)))
print("Test set score: {:.3f}".format(logreg.score(X_test, y_test)))
```

```
Training set score: 0.781
Test set score: 0.766
```

In [96]:

```
logreg001 = LogisticRegression(C=0.01).fit(X_train, y_train)
print("Training set accuracy: {:.3f}".format(logreg001.score(X_train, y_train)))
print("Test set accuracy: {:.3f}".format(logreg001.score(X_test, y_test)))
```

```
Training set accuracy: 0.762
Test set accuracy: 0.760
```

3. Random Forest

In [97]:
```
rf = RandomForestClassifier(n_estimators=100, random_state=0)
rf.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(rf.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(rf.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.786
```

In [98]:
```
rf1 = RandomForestClassifier(max_depth=3, n_estimators=100, random_state=0)
rf1.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(rf1.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(rf1.score(X_test, y_test)))
```

```
Accuracy on training set: 0.800
Accuracy on test set: 0.755
```

In [ ]:
```

```