Technological Institute of the Philippines	**Quezon City - Computer Engineering**		
Course Code:	**CPE 019**		
Code Title:	**Emerging Technologies 2 in CpE 1**	**2nd Semester**	**AY 2023-2024**
ACTIVITY NO.	**PRELIM PROJECT**		
Name	**Beato, Danica Marie L., Guevarra, Hans Angelo**		
Section	**CPE32S3**		
Date Performed:	**03/03/2024**		
Date Submitted:	**03/06/2024**		
Instructor:	**Engr. Roman Richard**		

- **Choose any dataset applicable for classification and/or prediction analysis problems.**
- **Show the application of the following algorithms:**

In [ ]:

```
# Import libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
diabetes = pd.read_csv("/content/diabetes.csv")
```

In [ ]:

```
diabetes
```

Out[ ]:

	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
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

In [ ]:

```
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
```

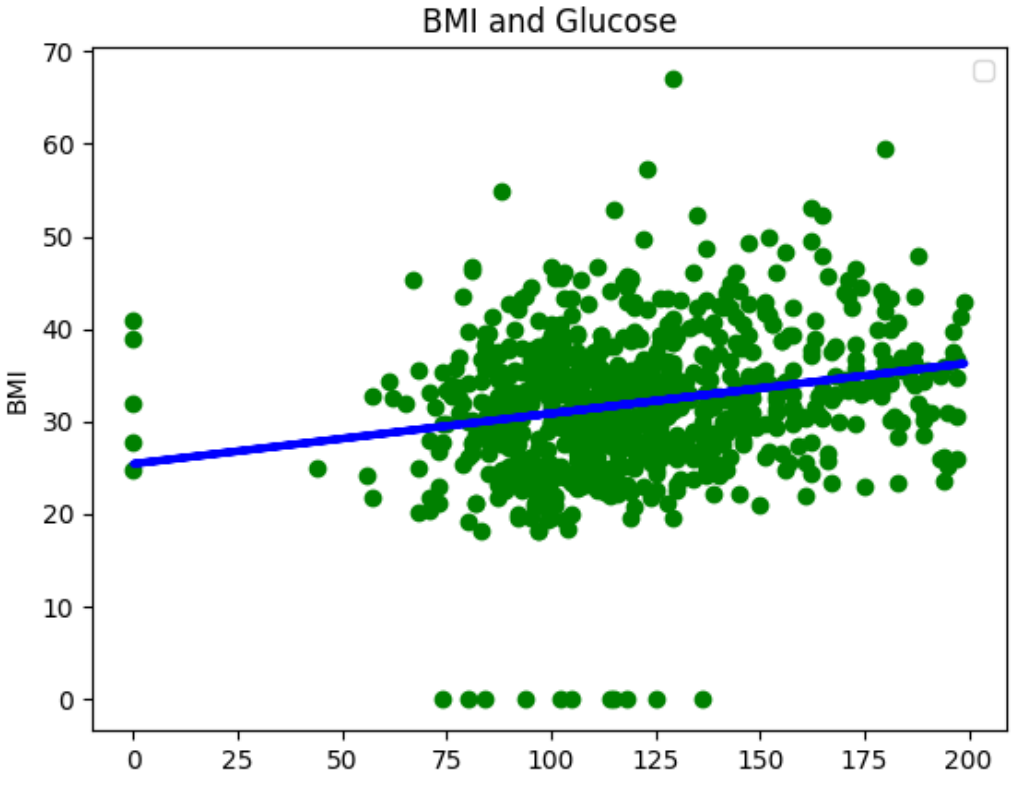
Linear Regression

Singular Linear Regression

In [ ]:

```
from sklearn.linear_model import LinearRegression

LR = LinearRegression()
LR.fit(diabetes[['Glucose']], diabetes['BMI'])
y_prediction = LR.predict(diabetes[['Glucose']])

plt.scatter(diabetes[['Glucose']], diabetes['BMI'], color='green')
plt.plot(diabetes[['Glucose']], y_prediction, color='blue', linewidth = 3)
plt.xlabel('Glucose')
plt.ylabel('BMI')
plt.title('BMI and Glucose')
plt.legend()
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artis
ts whose label start with an underscore are ignored when legend() is called with no argum
ent.
```



Glucose

Evaluation Report:

In order to evaluate the singular LR model, metrics such as mean squared error, R-squared, or coefficient of determination could be calculated. The generated plot shows a scatter of data points representing glucose levels and corresponding BMI values. A straight line is plotted through the points, illustrating the linear relationship between glucose and BMI predicted by the model. As the glucose levels increase, the model predicts a corresponding increase in BMI values.

Multiple Linear Regression

In [ ]:

```
X = diabetes[['Glucose', 'Age', 'BloodPressure']]
y = diabetes['Outcome']
```

In [ ]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state
= 0)
```

In [ ]:

```
from sklearn.linear_model import LinearRegression

MLr = LinearRegression()
MLr.fit(X_train, y_train)
```

Out[ ]:

▾ LinearRegression

LinearRegression()

In [ ]:

```
c = MLr.intercept_
c
```

Out[ ]:

```
-0.5804874989771283
```

In [ ]:

```
m = MLr.coef_
m
```

Out[ ]:

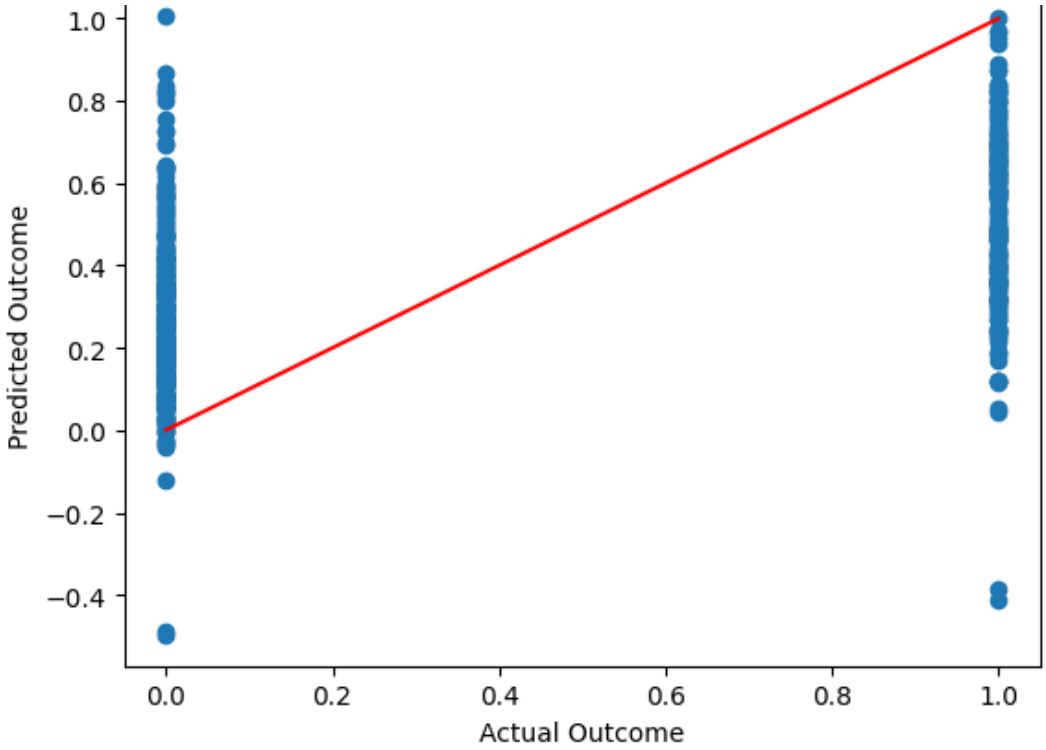
```
array([ 0.00648679,  0.00538734, -0.00040291])
```

In [ ]:

```
y_pred_train = MLr.predict(X_train)
```

In [ ]:

```
plt.scatter(y_train, y_pred_train)
plt.xlabel("Actual Outcome")
plt.ylabel("Predicted Outcome")
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], color='red', label=
'Regression Line')
plt.show()
```



Evaluation Report:

The generated multiple LR plot displays a scatter of data points representing age and glucose levels, with each point labeled according to the diabetes outcome. Additionally, the plot includes a decision boundary that separates the data points into different regions corresponding to the predicted outcomes. It visually assesses how well the model separates the data points into their respective classes based on age and glucose levels. The decision boundary indicates the model's classification boundaries.

Polynomial Linear Regression

In [ ]:

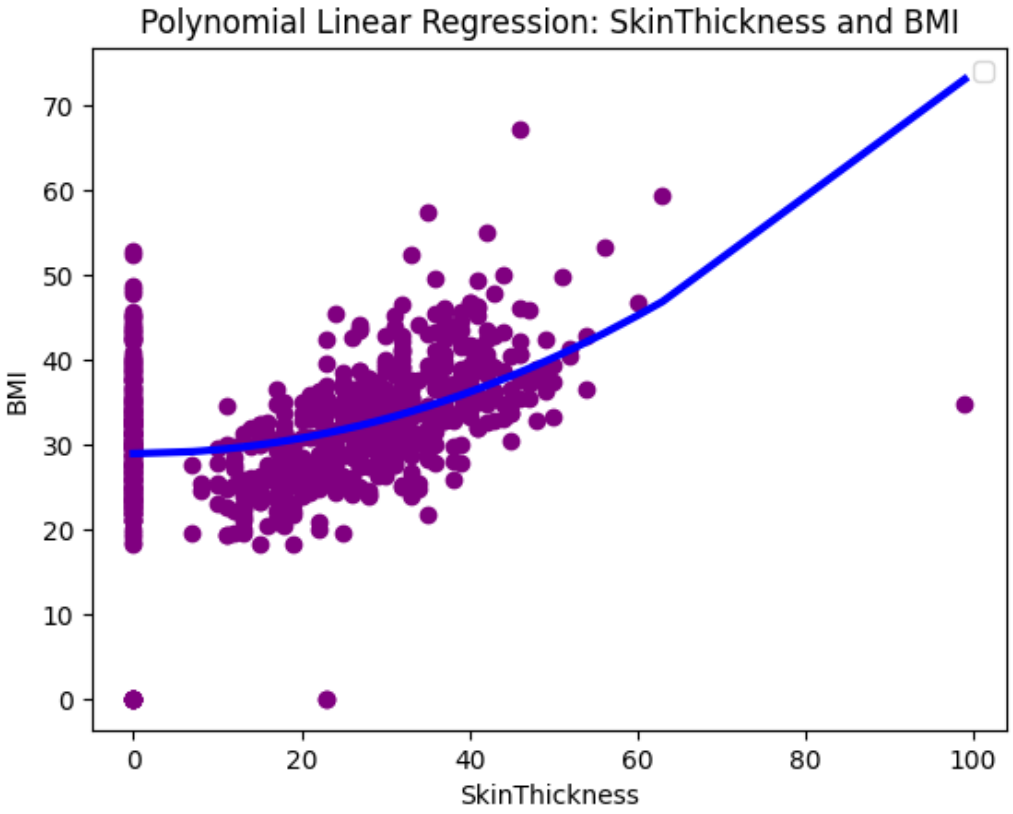
```
# Import libraries

from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
import seaborn as sns
```

In [ ]:

```
Xpoly = diabetes[['SkinThickness']]
ypoly = diabetes['BMI']


degree = 2

# Polynomial regression pipeline
pipeline = make_pipeline(PolynomialFeatures(degree), LinearRegression())
pipeline.fit(Xpoly, ypoly)

y_pred_poly = pipeline.predict(Xpoly)

# Sort the values for a smoother plot
sort_order = np.argsort(Xpoly.values.flatten())

plt.scatter(Xpoly, ypoly, color='purple')

# Regression line
plt.plot(Xpoly.iloc[sort_order], y_pred_poly[sort_order], color='blue', linewidth=3)
plt.xlabel('SkinThickness')
plt.ylabel('BMI')
plt.title('Polynomial Linear Regression: SkinThickness and BMI')
plt.legend()
```

```
plt.show()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artis
ts whose label start with an underscore are ignored when legend() is called with no argum
ent.
```



Evaluation Report:

The plot of the polynomial LR model depicts a scatter of data points representing skin thickness and BMI values, with a curved line illustrating the model's prediction of BMI based on skin thickness. This visualization highlights how the model captures the non-linear relationship between the two variables.

Logistic Regression

In [ ]:

```
# Import libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
```

In [ ]:

```
X1 = diabetes[['Age', 'Glucose']]
y1 = diabetes['Outcome']
```

In [ ]:

```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1)
```

In [ ]:

```
log_reg = LogisticRegression()
log_reg.fit(X1_train, y1_train)

y1_pred = log_reg.predict(X1_test)
```

In [ ]:

```
confusion_matrix(y1_test, y1_pred)
```

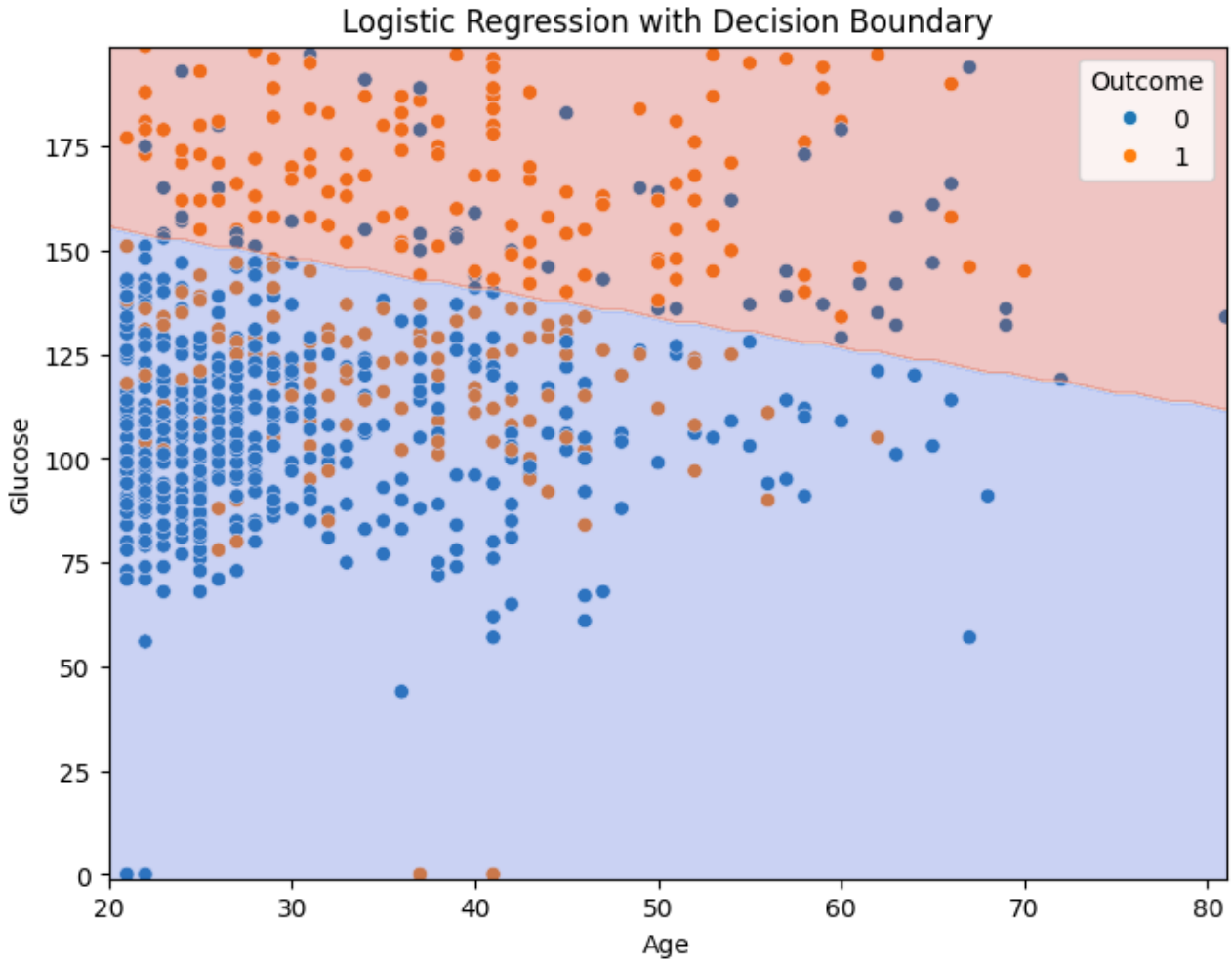
Out[ ]:

```
array([[105,  12],
       [ 37,  38]])
```

In [ ]:

```
x_min, x_max = X1.iloc[:, 0].min() - 1, X1.iloc[:, 0].max() + 1
y_min, y_max = X1.iloc[:, 1].min() - 1, X1.iloc[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max), np.arange(y_min, y_max))


Z = log_reg.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plotting the data points
plt.figure(figsize=(8, 6))
scatter = sns.scatterplot(x='Age', y='Glucose', hue='Outcome', data=diabetes, edgecolor=
'w')
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.3)

# Legend

plt.title('Logistic Regression with Decision Boundary')
plt.xlabel('Age')
plt.ylabel('Glucose')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have
valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
```



Evaluation Report:

The plot of the logistic regression model displays a scatter of data points representing age and glucose levels, with each point labeled according to the diabetes outcome. The model distinguishes between positive and negative diabetes outcomes based on age and glucose levels. In the context of of this dataset, a positive coefficient for age suggests that as age increases, the odds of having diabetes also increase. As we can see on

the plot, individuals with higher ages are more likely to have diabetes according to the logistic regression model. This observation supports the common understanding that age is a significant factor in the development of diabetes.

Decision Tree

In [ ]:

```
# Create an array object with the variable that will be the target for the model
y_target = diabetes["Outcome"].values
```

In [ ]:

```
# Create an array of the values that will be the input for the model
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI",
"DiabetesPedigreeFunction", "Age"]
X_input = diabetes[list(columns)].values
```

In [ ]:

```
# Create the learned model
from sklearn import tree
clf_train = tree.DecisionTreeClassifier(criterion="entropy", max_depth=3)
clf_train = clf_train.fit(X_input, y_target)
```

In [ ]:

```
# Evaluate the model
clf_train.score(X_input,y_target)
```

Out[ ]:

```
0.7734375
```

In [ ]:

```
# Create the intermediate file output
with open("/content/diabetes.csv", 'w') as f:
  f = tree.export_graphviz(clf_train, out_file=f, feature_names=columns)
```

In [ ]:

```
!apt-get install graphviz
```

```
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
graphviz is already the newest version (2.42.2-6).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
```

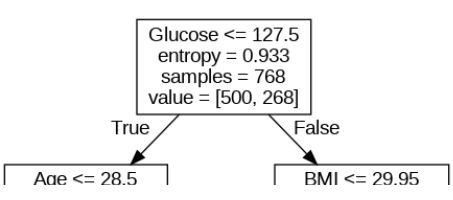
In [ ]:

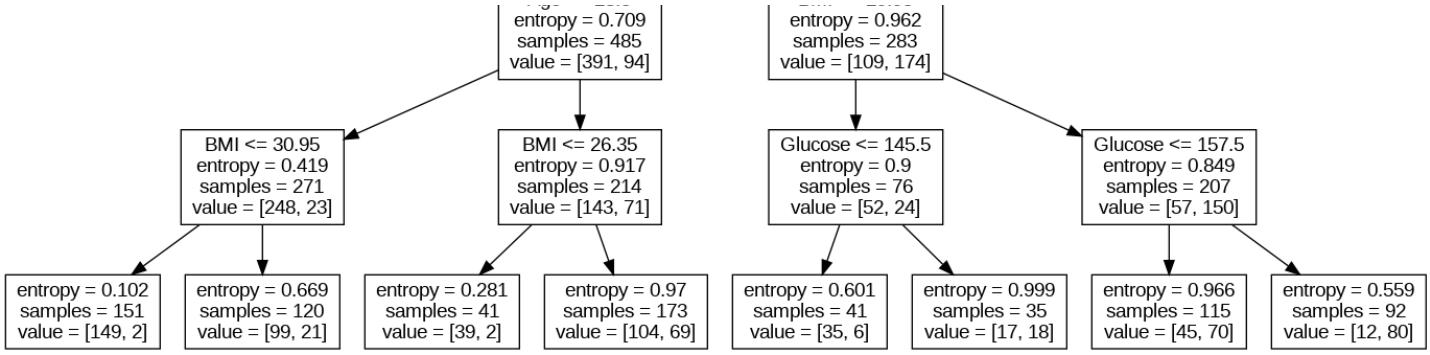
```
#run the Graphviz dot command to convert the .dot file to .png
!dot -Tpng /content/diabetes.csv -o .diabetes.png
```

In [ ]:

```
#import the Image module from the Ipython.display libary
from IPython.display import Image
#display the decison tree graphic
Image(".diabetes.png")
```

Out[ ]:





Evaluation Report:

The decision tree model achieves an accuracy score of approximately 77.34%, indicating its ability to correctly classify diabetes outcomes based on the provided features. While the decision tree's visualization provides insights into the model's decision-making process, the accuracy score quantifies its performance. With an accuracy of 77.34%, the model demonstrates decent predictive capability.

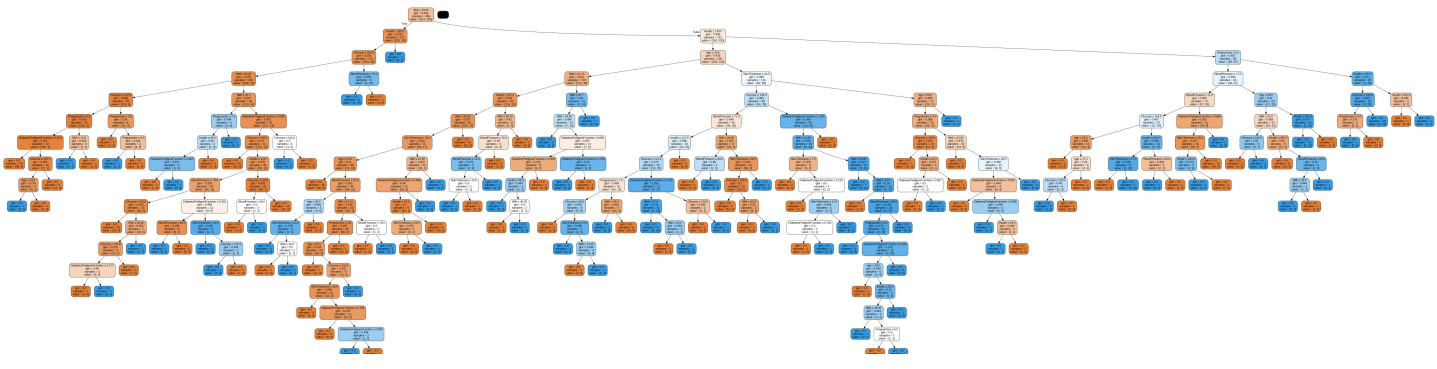
Random Forest

In [ ]:

```
# Previously created array objects of the target and input variables
y_target = diabetes["Outcome"].values
X_input = diabetes[columns].values

# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus
from sklearn import tree

# Create and train the Random Forest model
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_input, y_target)

# Model accuracy
rf_accuracy = rf_classifier.score(X_input, y_target)
print("Random Forest Model Accuracy:", rf_accuracy)

# Export the first decision tree in the Random Forest to an intermediate file output
with open("/content/diabetes.csv", 'w') as f:
    f = export_graphviz(rf_classifier.estimators_[0], out_file=f, feature_names=columns,
filled=True, rounded=True, special_characters=True)

# Convert the .dot file to .png
graph = pydotplus.graph_from_dot_file("/content/diabetes.csv")
graph.write_png("/content/diabetes_rf.png")

# Display the image of the first decision tree in the Random Forest
Image("/content/diabetes_rf.png")
```

```
Random Forest Model Accuracy: 1.0
```

Out[ ]:



Evaluation reports:

It's quite tricky as there are a lot of 'branches' in this model. The code was based on the decision tree model by importing 'RandomForestClassifier'. Overall, the Random Forest model demonstrates promising results with high accuracy and provides insights into feature importance and decision-making processes.