CHHOTUBHAI GOPALBHAI PATEL INSTITUTE OF TECHNOLOGY

Information Technology

Machine Intelligence (CE6001)

List of Experiments

Sr. No.	Title	Sign
1	To study various machine learning libraries like - Scipy, Sklearn, Keras, TensorFlow with their usage.	
2	Write a python program to perform Linear classification using AND and OR logic.	
3	Write a python program to perform multiclass classification on iris dataset.	
4	Write a python program to classify various types of from iris dataset using Support Vector Machine (SVM).	
5	Write a program to perform dimensionality reduction on Iris dataset using Principal Component Analysis (PCA).	
6	Write a program to implement K-means clustering on iris dataset.	
7	Write a program to apply a decision tree classifier on pima Indian diabetes dataset.	

Practical-1

Aim: To study various machine learning libraries like - Scipy, Sklearn, Keras, TensorFlow with their usage.

Description:

1. SciPy:

• Overview: SciPy is an open-source library used for scientific and technical computing. It builds on NumPy and provides a variety of numerical algorithms for optimization, integration, interpolation, eigenvalue problems, and more.

• Usage:

- Optimization (e.g., curve fitting)
- Signal processing
- Image processing

• Applications:

- Signal Processing: Used in fields like audio and telecommunications to filter signals and remove noise.
- Optimization: SciPy's optimization methods are used in financial modeling and resource allocation to find optimal solutions.
- Image Processing: It is used in medical imaging to enhance images (MRI, CT scans).
- Physics Simulations: Simulating complex systems in physics like heat transfer and wave propagation.

2. Sklearn:

• Overview: Scikit-learn is a powerful library for machine learning in Python. It provides simple and efficient tools for data mining and data analysis.

• Usage:

- Supervised learning (classification, regression)
- Unsupervised learning (clustering, dimensionality reduction)
- Model evaluation (cross-validation, performance metrics)
- Preprocessing data (scaling, encoding)

- Applications:
  - Spam Detection: Classifies emails as spam or non-spam using classification algorithms like Naive Bayes.
  - Customer Segmentation: Used in marketing to divide customers into different groups using clustering methods.
  - Predictive Maintenance: Helps in industries to predict machine failures based on sensor data using regression models.
  - Recommendation Systems: Implements collaborative filtering techniques to recommend products or content to users (like in e-commerce or streaming services).

3. Keras:

- Overview: Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It simplifies the process of building deep learning models.

- Usage:
  - Building neural networks (sequential and functional API)
  - Training, evaluating, and fine-tuning deep learning models
  - Handling Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), etc.

- Applications:
  - Image Classification: Widely used for classifying images into categories, such as recognizing handwritten digits or classifying objects in photos (e.g., MNIST, CIFAR datasets).
  - Natural Language Processing (NLP): Sentiment analysis, text classification, and language translation using Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) models.
  - Speech Recognition: Training neural networks to convert speech to text using deep learning techniques.
  - Healthcare: Keras is used for detecting diseases from medical images (e.g., cancer detection from CT scans).

4. TensorFlow:

- Overview: TensorFlow is an open-source library developed by Google for numerical computation and machine learning, with a focus on deep learning models. It is highly flexible and scalable for building neural networks.

- Usage:
    - Building large-scale neural networks
    - Model deployment in production environments
    - Running on GPUs for faster computation
    - Supports reinforcement learning, generative models, etc.

- Applications:
    - Self-Driving Cars: TensorFlow is used in autonomous vehicle systems to identify objects like pedestrians, traffic signs, and other vehicles.
    - Voice Assistants: Powers virtual assistants like Google Assistant by enabling natural language understanding and speech synthesis.
    - Fraud Detection: TensorFlow is employed in banking and financial systems to detect fraudulent transactions using deep neural networks.
    - Robotics: TensorFlow is used for enabling robots to process visual and sensor data to perform tasks autonomously, such as in industrial automation.

Practical-2

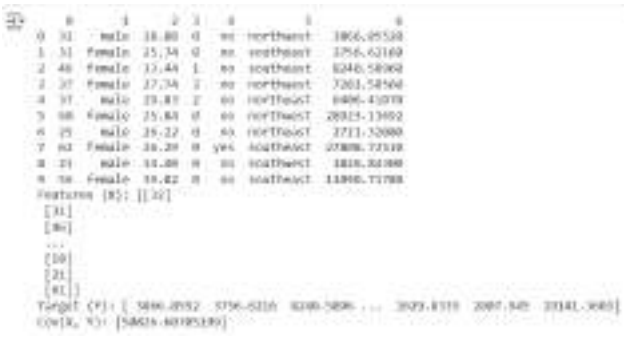
Aim: Write a python program to perform Linear classification using AND and OR logic.

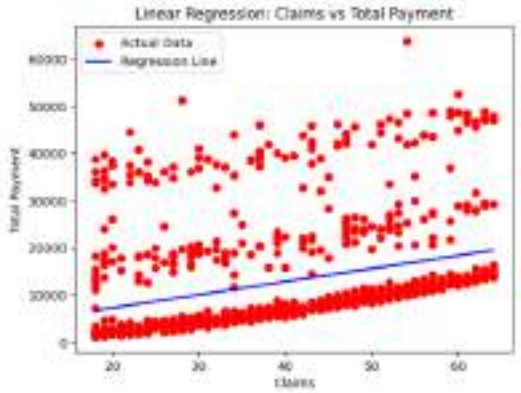
Code:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression

# Load the dataset
dataset = pd.read_csv('insurance.csv', skiprows=5, header=None)
print(dataset.head(10))

# Prepare the data
X = dataset.iloc[:, 0].values.reshape(-1, 1)  # Reshaping for sklearn
Y = dataset.iloc[:, -1].values  # Assuming last column is the target

print("Features (X):", X)
print("Target (Y):", Y)

# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=1)

# Calculate mean and variance
X_mean = np.mean(X)
Y_mean = np.mean(Y)

X_variance = np.var(X)
Y_variance = np.var(Y)

def covariance(X, Y):
    x_mean = np.mean(X)
    y_mean = np.mean(Y)
    covar = 0.0
    for i in range(len(X)):
        covar += (X[i] - x_mean) * (Y[i] - y_mean)
```

```
    return covar / len(X)

# Calculate covariance
covar_xy = covariance(X, Y)
print(f'Cov(X, Y): {covar_xy}')

# Linear regression model
regressor = LinearRegression()
regressor.fit(X_train, Y_train)

# Predictions
y_pred = regressor.predict(X_train)

# Plotting the results
plt.scatter(X_train, Y_train, color='red', label='Actual Data')
plt.plot(X_train, y_pred, color='blue', label='Regression Line')
plt.xlabel('Claims')
plt.ylabel('Total Payment')
plt.title('Linear Regression: Claims vs Total Payment')
plt.legend()
plt.show()
```

Output:





Practical-3

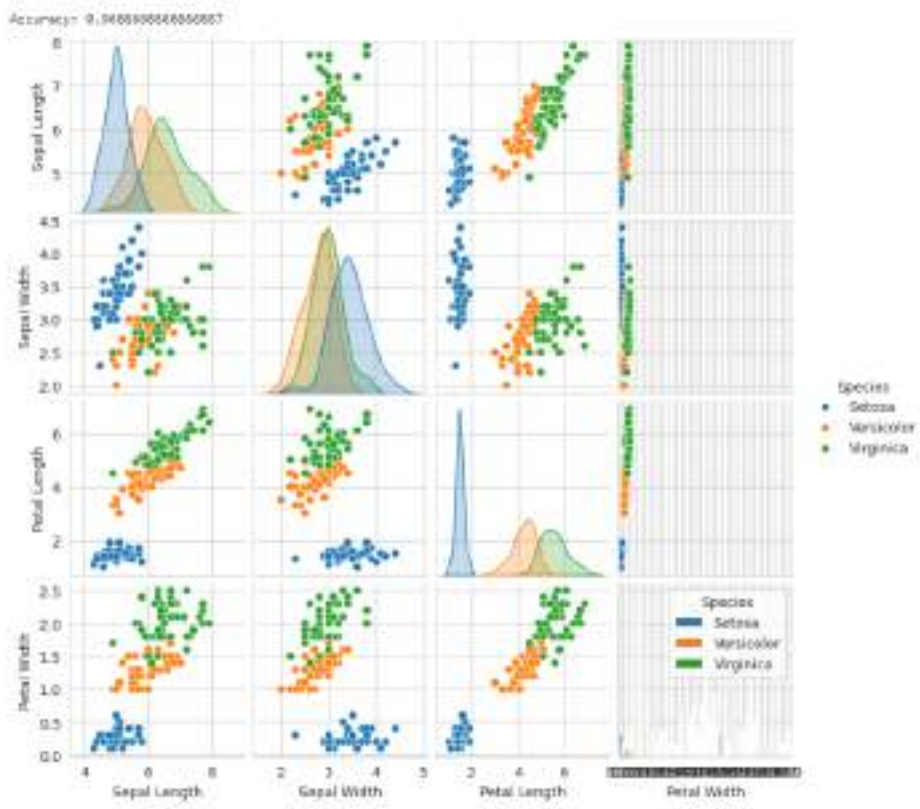
Aim: Write a python program to perform multiclass classification on iris dataset.

Code:

```
# Data imports
import numpy as np
import pandas as pd
from pandas import Series, DataFrame
# Plot imports
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
from sklearn import linear_model
from sklearn.datasets import load_iris
# Load the iris dataset
iris = load_iris()
X = iris.data
Y = iris.target
print(iris.DESCR)
# Create dataframes for visualizations
iris_data = DataFrame(X, columns=['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'])
iris_target = DataFrame(Y, columns=['Species'])
# Function to label flowers
def flower(num):
    if num == 0:
        return 'Setosa'
    elif num == 1:
        return 'Versicolor'
    else:
        return 'Virginica'

# Label flowers and combine dataframes
iris_target['Species'] = iris_target['Species'].apply(flower)
iris = pd.concat([iris_data, iris_target], axis=1)
# Visualizations
sns.pairplot(iris, hue='Species', height=2)
```

```
sns.countplot(x='Petal Length', hue='Species', data=iris)
# Train the model
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
log_reg = LogisticRegression()
# Split the dataset into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=3)
# Fit the logistic regression model
log_reg.fit(X_train, Y_train)
# Test accuracy
from sklearn import metrics
Y_pred = log_reg.predict(X_test)
print("Accuracy:", metrics.accuracy_score(Y_test, Y_pred))
```

Output:



Practical-4

Aim: Write a python program to classify various types of from iris dataset using Support Vector Machine (SVM).

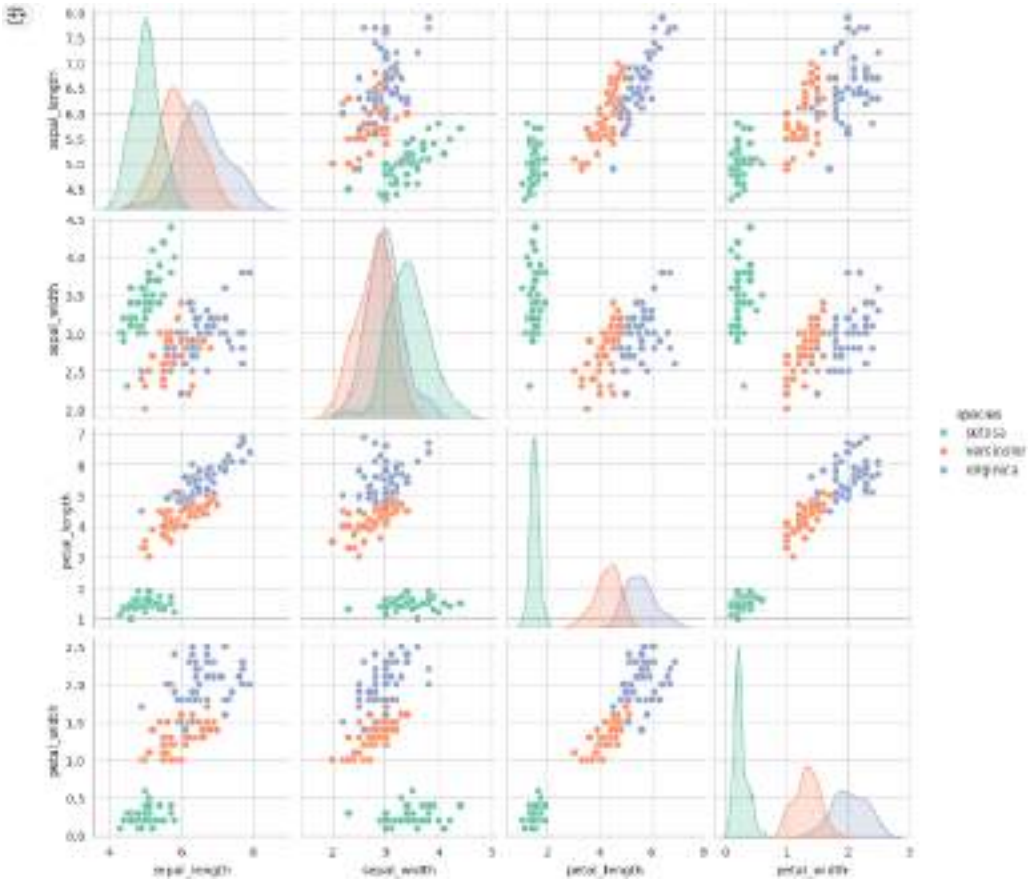
Code:

```
# Import necessary libraries
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
# Ensure inline plotting for Jupyter notebooks
%matplotlib inline
# Check the available datasets in the input directory
# print(os.listdir("../input"))
# Import the dataset using Pandas
iris = pd.read_csv('/content/iris.csv')
# Display the first few rows of the dataset
print(iris.head())
# Visualize pairplot for the dataset
sns.pairplot(data=iris, hue='species', palette='Set2')
plt.show()
# Split the dataset into features and target variable
X = iris.iloc[:, :-1]  # Features (all columns except the last)
y = iris.iloc[:, -1]   # Target (the last column)
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)

# Initialize the SVC model
model = SVC()
# Fit the model to the training data
model.fit(X_train, y_train)
# Make predictions on the test data
pred = model.predict(X_test)
# Generate and print the confusion matrix
```

```
print(confusion_matrix(y_test, pred))
# Generate and print the classification report
print(classification_report(y_test, pred))
```

Output:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa




```
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

Practical-5

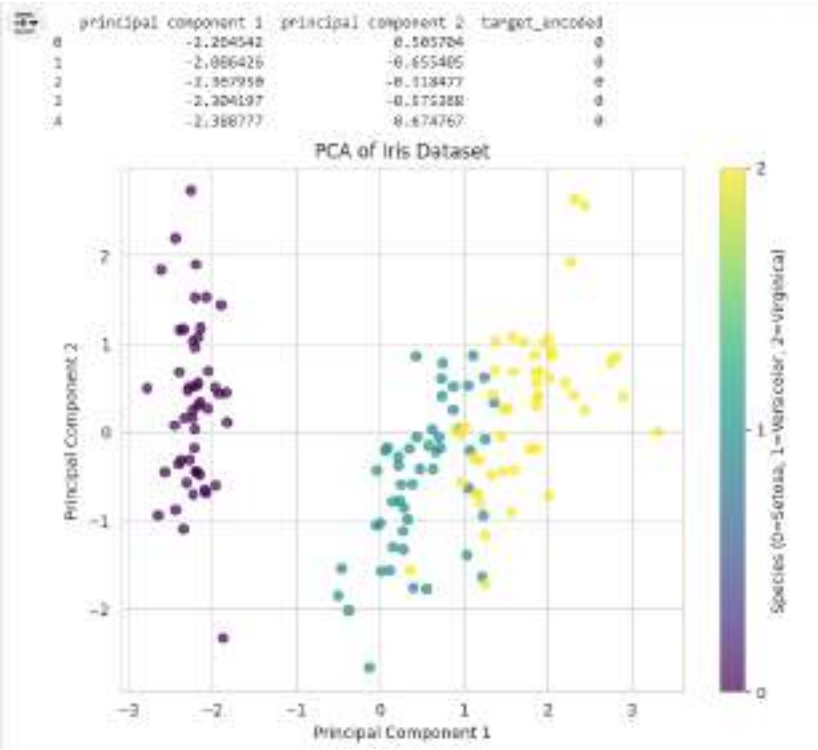
Aim: Write a program to perform dimensionality reduction on Iris dataset using Principal Component Analysis (PCA).

Code:

```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
# Load the dataset into a Pandas DataFrame
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
df = pd.read_csv(url, names=['sepal length', 'sepal width', 'petal length', 'petal width', 'target'])
# Separating out the features and the target
features = ['sepal length', 'sepal width', 'petal length', 'petal width']
x = df.loc[:, features].values  # Features
y = df.loc[:, ['target']].values.ravel()  # Target, flattened to 1D
# Standardizing the features
x = StandardScaler().fit_transform(x)
# Performing PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
# Creating a DataFrame with the principal components
principalDf = pd.DataFrame(data=principalComponents, columns=['principal component 1',
'principal component 2'])
# Converting target species to numerical values using LabelEncoder
label_encoder = LabelEncoder()
df['target_encoded'] = label_encoder.fit_transform(df['target'])
# Concatenating the principal components with the target variable
finalDf = pd.concat([principalDf, df[['target_encoded']]], axis=1)
# Displaying the first few rows of the final DataFrame
print(finalDf.head())
# Optional: Visualizing the PCA results
plt.figure(figsize=(8, 6))
scatter = plt.scatter(finalDf['principal component 1'], finalDf['principal component 2'],
                      c=finalDf['target_encoded'], cmap='viridis', alpha=0.7)
plt.title('PCA of Iris Dataset')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
```

```
plt.colorbar(scatter, ticks=[0, 1, 2], label='Species (0=Setosa, 1=Versicolor, 2=Virginica)')
plt.show()
```

Output:



Practical-6

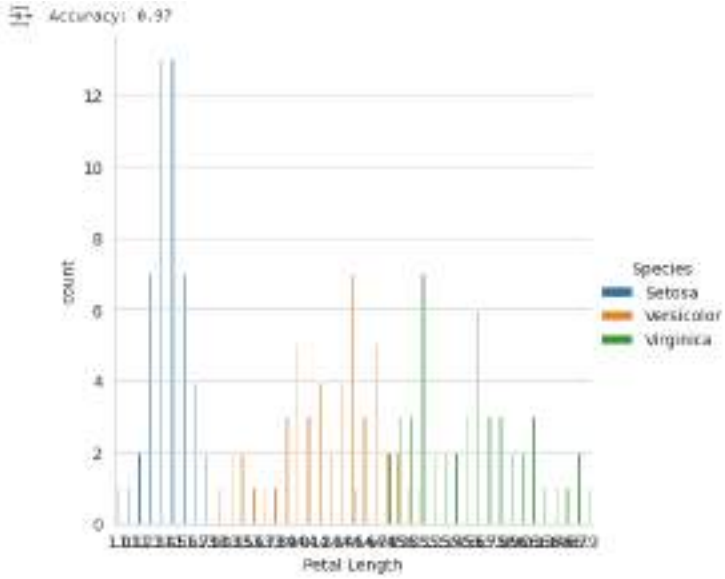
Aim: Write a program to implement K-means clustering on iris dataset.

Code:

```
# Data imports
import numpy as np
import pandas as pd
from pandas import DataFrame
# Plot imports
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('whitegrid')
%matplotlib inline
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn import metrics
# Load the iris dataset
iris = load_iris()
X = iris.data
Y = iris.target
# Create DataFrames for visualizations
iris_data = DataFrame(X, columns=['Sepal Length', 'Sepal Width', 'Petal Length', 'Petal Width'])
iris_target = DataFrame(Y, columns=['Species'])
# Function to label the species
def flower(num):
    if num == 0:
        return 'Setosa'
    elif num == 1:
        return 'Versicolor'
    else:
        return 'Virginica'

# Label flowers
iris_target['Species'] = iris_target['Species'].apply(flower)
# Combine DataFrames
iris = pd.concat([iris_data, iris_target], axis=1)
# Visualize the data
```

```
sns.catplot(x='Petal Length', data=iris, hue='Species', height=5, kind='count')
# Train the logistic regression model
log_reg = LogisticRegression()
 Train-test split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.4, random_state=3)
log_reg.fit(X_train, Y_train)
# Test accuracy
Y_pred = log_reg.predict(X_test)
accuracy = metrics.accuracy_score(Y_test, Y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

Output:



Practical-7

Aim: Write a program to apply a decision tree classifier on pima Indian diabetes dataset.

Code:

```
# Import necessary libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
import pydot
from sklearn.tree import export_graphviz
from IPython.display import Image
from six import StringIO
# Load dataset
df = pd.read_csv('/content/diabetes.csv')  # Adjust this path as necessary
print("Dataset Head:\n", df.head())
 Feature variables
x = df.drop(['Outcome'], axis=1)
print("\nFeature Variables:\n", x.head())
# Target variable
y = df['Outcome']
print("\nTarget Variable:\n", y.head())
# Split the dataset into training and test sets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=1)
# Create Decision Tree classifier object
model = DecisionTreeClassifier()
# Train Decision Tree Classifier
model.fit(x_train, y_train)
# Predict the response for the test dataset
y_pred = model.predict(x_test)
# Evaluation using Accuracy score
print("\nAccuracy:", metrics.accuracy_score(y_test, y_pred) * 100)
# Evaluation using Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("\nConfusion Matrix:\n", conf_matrix)
# Evaluation using Classification report
```

```
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Checking prediction value
sample_prediction = model.predict([[6, 148, 72, 35, 0, 33.6, 0.627, 50]])
print("\nPrediction for input [6, 148, 72, 35, 0, 33.6, 0.627, 50]:", sample_prediction)
# Visualizing Decision Tree
dot_data = StringIO()
export_graphviz(model, out_file=dot_data, filled=True, rounded=True,
                special_characters=True, feature_names=x.columns, class_names=['0', '1'])
graph = pydot.graph_from_dot_data(dot_data.getvalue())[0]
graph.write_png('diabetes_tree.png')
# Display the decision tree image
Image(graph.create_png())
```

Output: