Ex. No. 5

<u>Date:</u>

REGRESSION MODEL

Aim:

To write a Python program to build Regression models

Algorithm:

Step 1. Import necessary libraries: numpy, pandas, matplotlib.pyplot, LinearRegression, mean_squared_error, and r2_score.
Step 2. Create a numpy array for waist and weight values and store them in separate variables.
Step 3. Create a pandas DataFrame with waist and weight columns using the numpy arrays.
Step 4. Extract input (X) and output (y) variables from the DataFrame.
Step 5. Create an instance of LinearRegression model.
Step 6. Fit the LinearRegression model to the input and output variables.
Step 7. Create a new DataFrame with a single value of waist.
Step 8. Use the predict() method of the LinearRegression model to predict the weight for the new waist value.
Step 9. Calculate the mean squared error and R-squared values using mean_squared_error() and r2_score() functions respectively.
Step 10. Plot the actual and predicted values using matplotlib.pyplot.scatter() and matplotlib.pyplot.plot() functions.

Program:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# import sample data using pandas
waist = np.array([70, 71, 72, 73, 74, 75, 76, 77, 78, 79])
weight = np.array([55, 57, 59, 61, 63, 65, 67, 69, 71, 73])
data = pd.DataFrame({'waist': waist, 'weight': weight})

# extract input and output variables
X = data[['waist']]
y = data['weight']

# fit a linear regression model
model = LinearRegression()
model.fit(X, y)
```

```
# make predictions on new data
new_data = pd.DataFrame({'waist': [80]})
predicted_weight = model.predict(new_data[['waist']])
print("Predicted weight for new waist value:", int(predicted_weight))

#calculate MSE and R-squared
y_pred = model.predict(X)
mse = mean_squared_error(y, y_pred)
print('Mean Squared Error:', mse)
r2 = r2_score(y, y_pred)
print('R-squared:', r2)

# plot the actual and predicted values
plt.scatter(X, y, marker='*', edgecolors='g')
plt.scatter(new_data, predicted_weight, marker='*', edgecolors='r')
plt.plot(X, y_pred, color='y')
plt.xlabel('Waist (cm)')
plt.ylabel('Weight (kg)')
plt.title('Linear Regression Model')
plt.show()
```

Viva questions:

1. What is a regression model?
2. What are the different types of regression models?
3. How do you determine which predictor variables to include in a regression model?
4. What is the difference between simple linear regression and multiple linear regression?
5. What are some common challenges in regression analysis and how can they be overcome?

Result:

Thus the Python program to build a simple linear Regression model was developed successfully.

Ex. No. 6 **DECISION TREE AND RANDOM FOREST**

Date:

Aim:

To write a Python program to build decision tree and random forest.

Algorithm:

Step 1. Import necessary libraries: numpy, matplotlib, seaborn, pandas, train_test_split, LabelEncoder, DecisionTreeClassifier, plot_tree, and RandomForestClassifier.
Step 2. Read the data from 'flowers.csv' into a pandas DataFrame.
Step 3. Extract the features into an array X, and the target variable into an array y.
Step 4. Encode the target variable using the LabelEncoder.
Step 5. Split the data into training and testing sets using train_test_split function.
Step 6. Create a DecisionTreeClassifier object, fit the model to the training data, and visualize the decision tree using plot_tree.
Step 7. Create a RandomForestClassifier object with 100 estimators, fit the model to the training data, and visualize the random forest by displaying 6 trees.
Step 8. Print the accuracy of the decision tree and random forest models using the score method on the test data.

Program:

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier

# read the data
data = pd.read_csv('flowers.csv')
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# encode the labels
le = LabelEncoder()
y = le.fit_transform(y)

# split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# create and fit a decision tree model
tree = DecisionTreeClassifier().fit(X_train, y_train)
```

```
# visualize the decision tree
plt.figure(figsize=(10,6))
plot_tree(tree, filled=True)
plt.title("Decision Tree")
plt.show()

# create and fit a random forest model
rf = RandomForestClassifier(n_estimators=100, random_state=0).fit(X_train, y_train)

# visualize the random forest
plt.figure(figsize=(20,12))
for i, tree_in_forest in enumerate(rf.estimators_[:6]):
    plt.subplot(2, 3, i+1)
    plt.axis('off')
    plot_tree(tree_in_forest, filled=True, rounded=True)
    plt.title("Tree " + str(i+1))
plt.suptitle("Random Forest")
plt.show()

# calculate and print the accuracy of decision tree and random forest
print("Accuracy of decision tree: {:.2f}".format(tree.score(X_test, y_test)))
print("Accuracy of random forest: {:.2f}".format(rf.score(X_test, y_test)))
```

Sample flowers.csv

```
Sepal_length,Sepal_width,Petal_length,Petal_width,Flower
4.6,3.2,1.4,0.2,Rose
5.3,3.7,1.5,0.2,Rose
5,3.3,1.4,0.2,Rose
7,3.2,4.7,1.4,Jasmin
6.4,3.2,4.5,1.5,Jasmin
7.1,3,5.9,2.1,Lotus
6.3,2.9,5.6,1.8,Lotus
```

Viva Questions:

1. What is the difference between a decision tree and a random forest?
2. How do you determine the best split at each node of a decision tree?
3. How do you prevent overfitting when building a decision tree?
4. How does the number of trees in a random forest affect the accuracy and performance of the model?
5. Can you explain how feature importance is calculated in a random forest model?

Result:

Thus the Python program to build decision tree and random forest was developed successfully.