**Ex No: 7 a) BUILD LINEAR REGRESSION MODELS**

# Program:

import pandas as pd

import statsmodels.api as sm

data = pd.read\_csv(“pima\_diabetes.csv")

#create correlation matrix

data.corr()

#Bivariate Analysis of Glucose-Insulin features

#define response variable 1

y1 = data['Glucose']

#define explanatory variable 1

x1 = data[['Insulin']]

#add constant to predictor variables

x1 = sm.add\_constant(x1)

#fit linear regression model

model1 = sm.OLS(y1, x1).fit()

#view model summary print(model1.summary())

#Bivariate Analysis of Age-Pregnancies features

#define response variable 2

y2 = data['Age']

#define explanatory variable 2

x2 = data['Pregnancies']

#add a constant to predictor variables

x2 = sm.add\_constant(x2)

#fit linear regression model

model2 = sm.OLS(y2, x2).fit()

#view model summary print(model2.summary())

#Bivariate Analysis of SkinThickness-BMI features

#define response variable 3

y3 = data['SkinThickness']

#define explanatory variable 3

x3 = data[['BMI']]

#add constant to predictor variables

x3 = sm.add\_constant(x3)

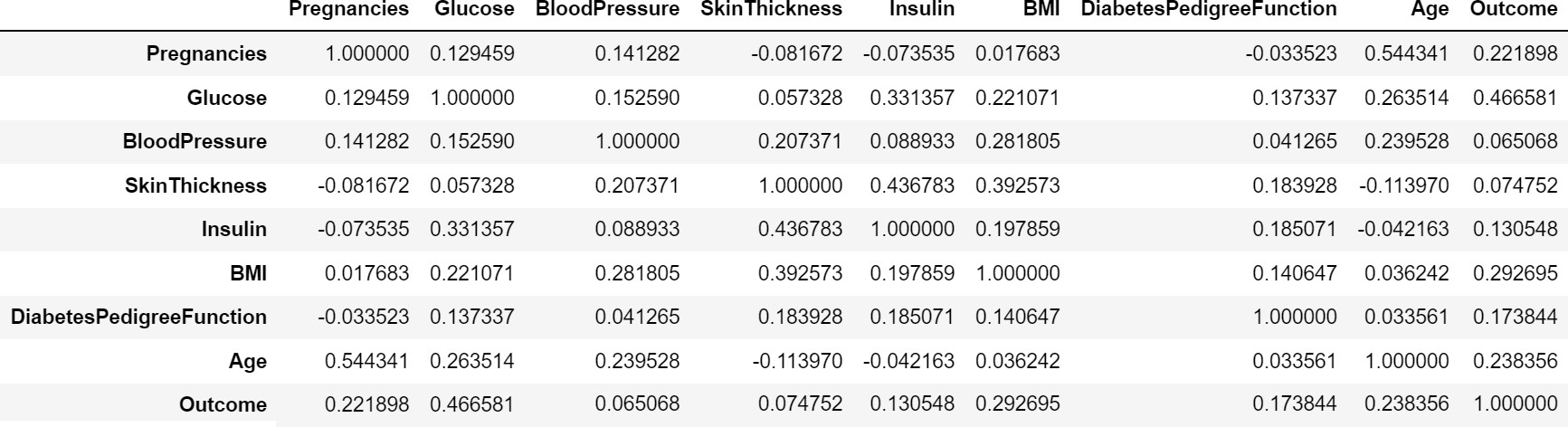
#fit linear regression model

Model3 = sm.OLS(y3, x3).fit()

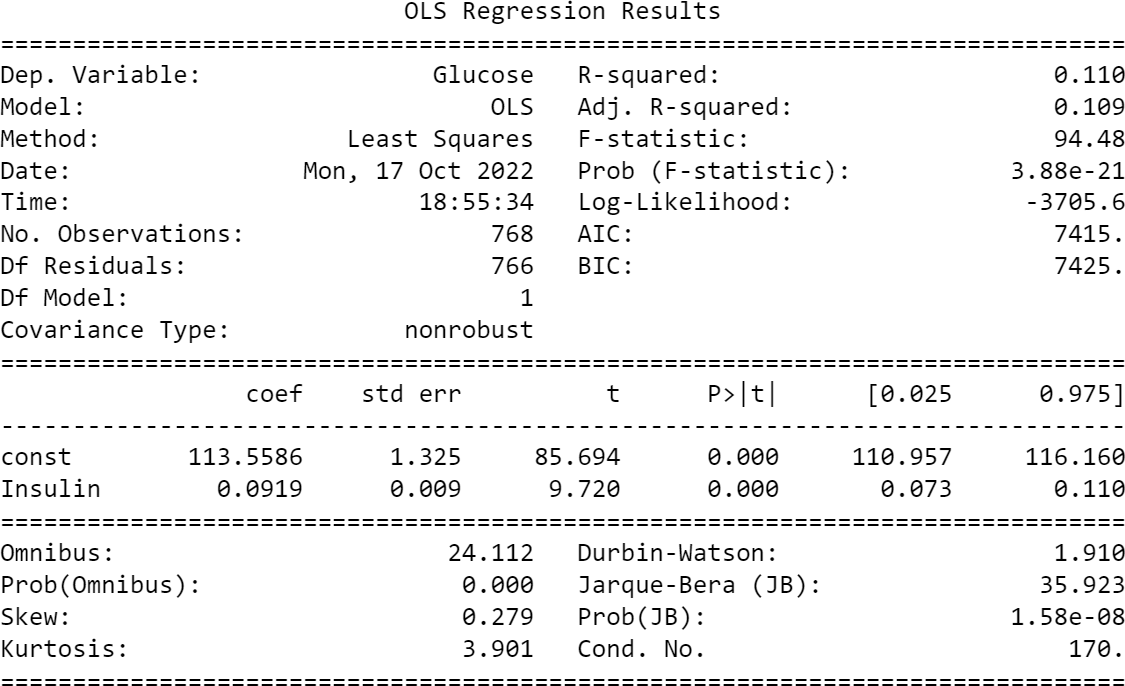
#view model summary print(model3.summary())

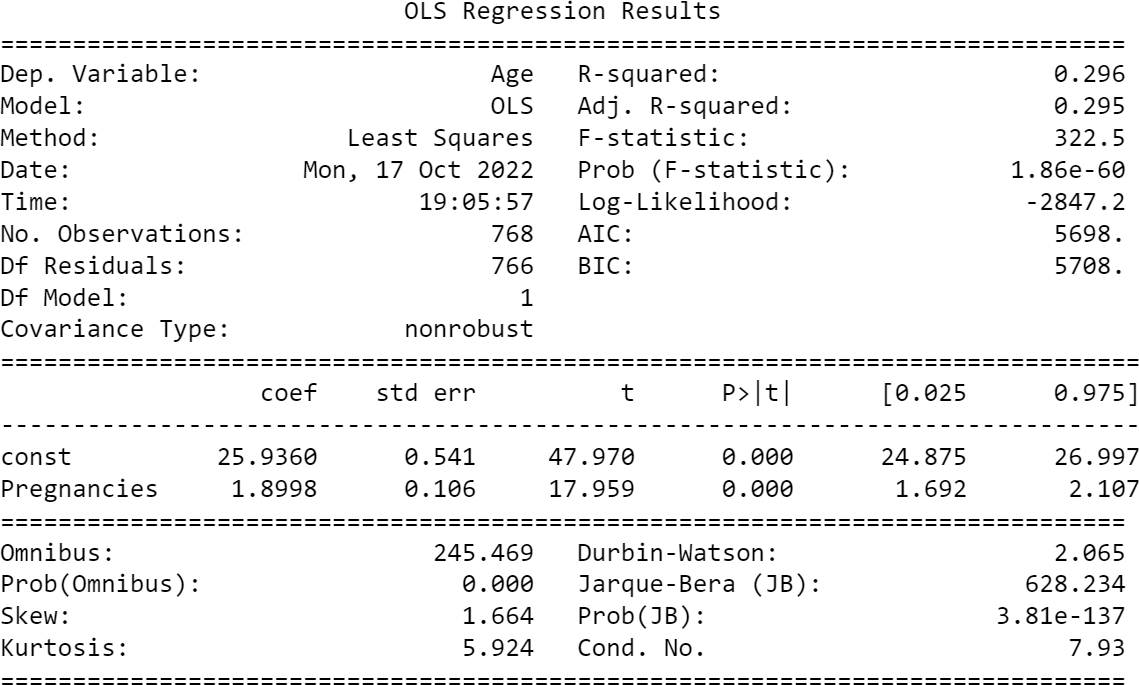
# Output:

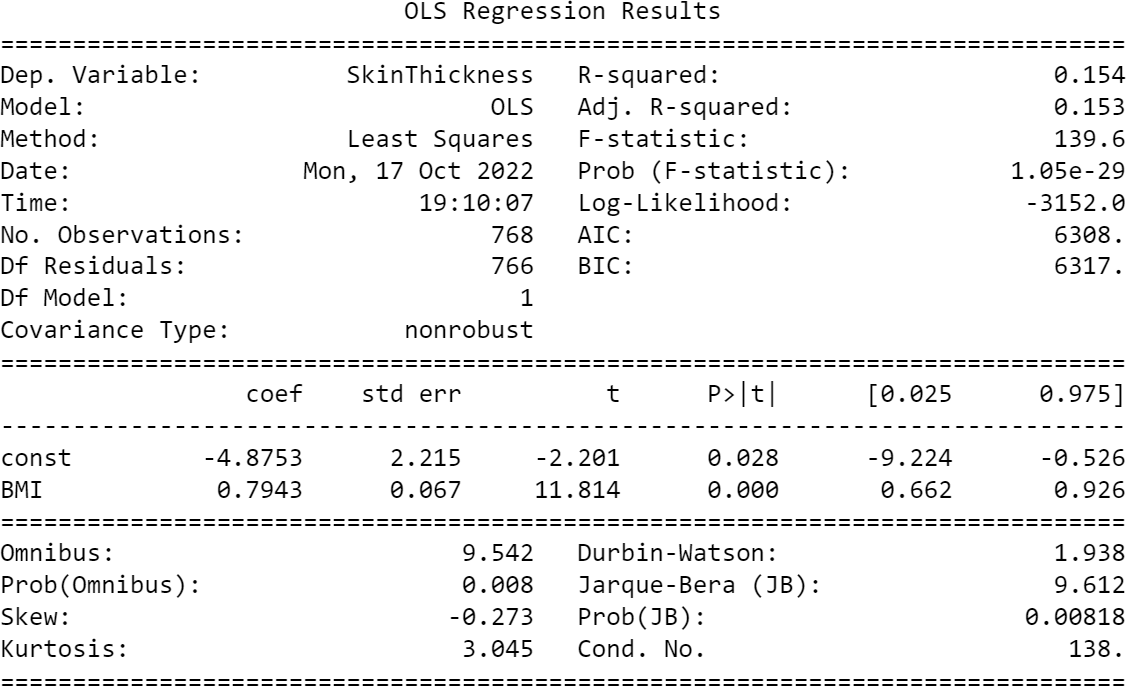
1. **Correlation Matrix**



1. **Bivariate Analysis of Glucose-Insulin features**



1. **Bivariate Analysis of Age-Pregnancies features**
2. **Bivariate Analysis of SkinThickness-BMI features**



**Ex No: 7 b) BUILD LOGISTIC REGRESSION MODELS**

# Program:

# importing libraries

import statsmodels.api as sm

import pandas as pd

# loading the training dataset

data = pd.read\_csv('pima\_diabetes.csv', index\_col = 0)

# defining the dependent and independent variables

Xtrain = data[['Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction','Age']]

ytrain = data[['Outcome']]

# building the model and fitting the data

log\_reg = sm.Logit(ytrain, Xtrain).fit()

# printing the summary table print(log\_reg.summary())

# Output:

**Ex No: 7 c) BUILD DECISION TREES**

# Program:

import pandas

from sklearn import tree

from sklearn.tree import DecisionTreeClassifier

df = pandas.read\_csv("data.csv")

print("Input:") print(df.head(5))

d = {'UK':0,'USA':1,'N':2}

df['Nationality'] = df['Nationality']. map(d)

d = {'YES':1, 'NO':0}

df['Go'] = df['Go']. map(d) print("Transformed Data:") print(df.head(5))

features = ['Age','Experience','Rank','Nationality']

X = df[features]

y = df['Go']

dtree = DecisionTreeClassifier()

dtree = dtree.fit(X,y) print(dtree.predict([[40,10,6,1]]))

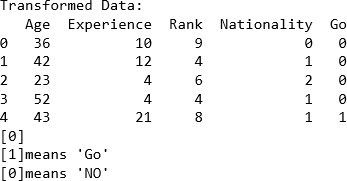
print("[1]means 'Go'")

print("[0]means 'NO'")

# DATA SET: (data.csv)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Age** | **Experience** | **Rank** | **Nationality** | **Go** |
| 36 | 10 | 9 | UK | NO |
| 42 | 12 | 4 | USA | NO |
| 23 | 4 | 6 | N | NO |
| 52 | 4 | 4 | USA | NO |
| 43 | 21 | 8 | USA | YES |

**Output:**



# Ex No: 7 d) BUILD RANDOM FORESTS

**Program:**

# Pandas is used for data manipulation import pandas as pd

# Read in data and display first 5 rows features = pd.read\_csv('temps.csv') features.head(5)

print('The shape of our features is:', features.shape)

# Descriptive statistics for each column features.describe()

# One-hot encode the data using pandas get\_dummies features = pd.get\_dummies(features)

# Display the first 5 rows of the last 12 columns features.iloc[:,5:].head(5)

import numpy as np

# Labels are the values we want to predict labels = np.array(features['actual'])

# Remove the labels from the features # axis 1 refers to the columns

features= features.drop('actual', axis = 1) # Saving feature names for later use feature\_list = list(features.columns)

# Convert to numpy array features = np.array(features)

# Using Skicit-learn to split data into training and testing sets from sklearn.model\_selection import train\_test\_split

# Split the data into training and testing sets

train\_features, test\_features, train\_labels, test\_labels = train\_test\_split(features, labels, test\_size = 0.25, random\_state = 42)

print('Training Features Shape:', train\_features.shape) print('Training Labels Shape:', train\_labels.shape) print('Testing Features Shape:', test\_features.shape) print('Testing Labels Shape:', test\_labels.shape)

# Import the model we are using

from sklearn.ensemble import RandomForestRegressor # Limit depth of tree to 3 levels

rf\_small = RandomForestRegressor(n\_estimators=10, max\_depth = 3) # Train the model on training data

rf\_small.fit(train\_features, train\_labels)

# Extract the small tree

tree\_small = rf\_small.estimators\_[5]

# Save the tree as a png image

export\_graphviz(tree\_small, out\_file = 'small\_tree.dot', feature\_names = feature\_list, rounded = True, precision = 1)

(graph, ) = ydot.graph\_from\_dot\_file('small\_tree.dot') graph.write\_png('small\_tree.png');

# Use the forest's predict method on the test data predictions = f\_small.predict(test\_features)

# Calculate the absolute errors

errors = abs(predictions - test\_labels)

# Print out the mean absolute error (mae)

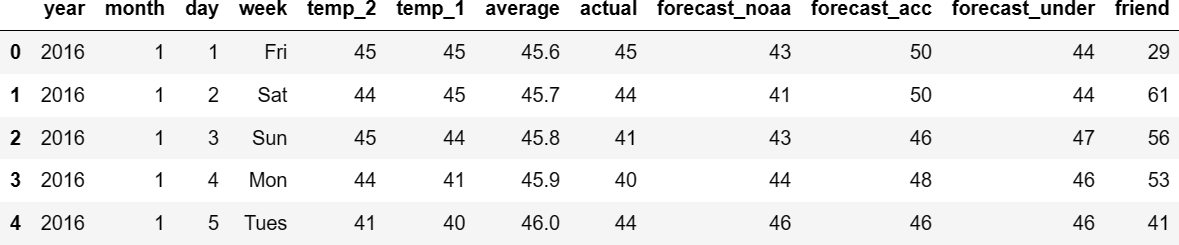
print('Mean Absolute Error:', round(np.mean(errors), 2), 'degrees.')

# Calculate mean absolute percentage error (MAPE) mape = 100 \* (errors / test\_labels)

# Calculate and display accuracy accuracy = 100 - np.mean(mape)

print('Accuracy:', round(accuracy, 2), '%.')

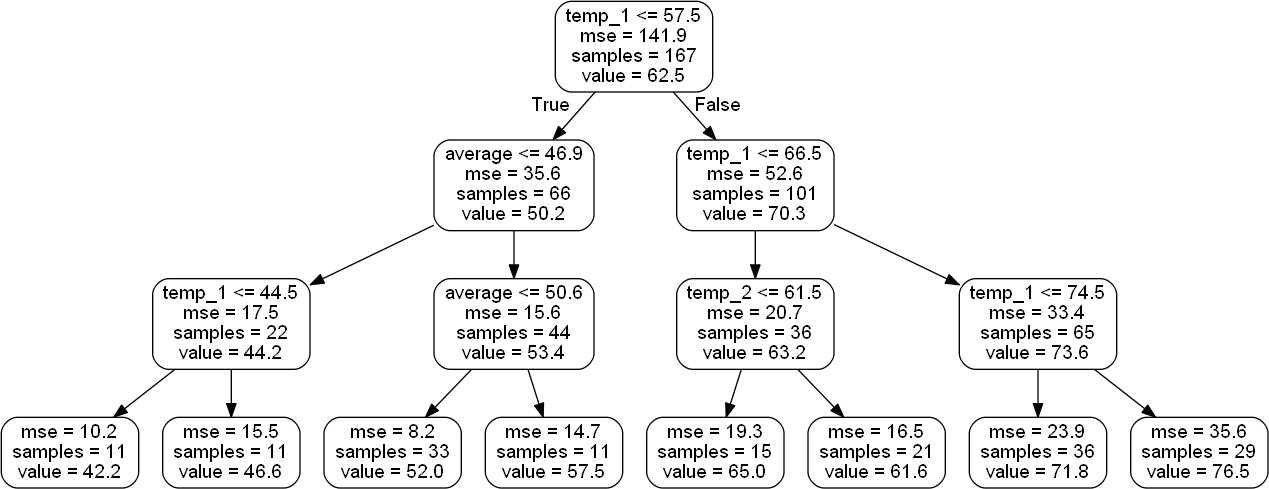
# Output:



The shape of our features is: (348, 12)

Training Features Shape: (261, 17) Training Labels Shape: (261,) Testing Features Shape: (87, 17) Testing Labels Shape: (87,)

RandomForestRegressor(max\_depth=3, n\_estimators=10)



Mean Absolute Error: 4.0 degrees. Accuracy: 93.73 %.

# Ex No: 7 e) BUILD SVM MODELS

**Program:**

import pandas

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix

data = pandas.read\_csv("vector.csv") print("Input: ")

print(data.head(10))

training\_set, test\_set = train\_test\_split(data, test\_size = 0.3, random\_state=1)

x\_train = training\_set.iloc[:,0:2].values

y\_train = training\_set.iloc[:,2].values

x\_test = test\_set.iloc[:,0:2].values

y\_test = test\_set.iloc[:,2].values

classifier = SVC(kernel='linear', random\_state=1) classifier.fit(x\_train, y\_train)

y\_pred = classifier.predict(x\_test) test\_set["prediction"] = y\_pred

print("Output")

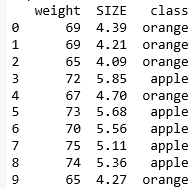
print(test\_set)

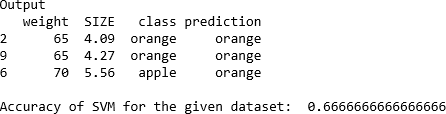
cm = confusion\_matrix(y\_test, y\_pred)

accuracy = float(cm.diagonal().sum()/len(y\_test))

print("\nAccuracy of SVM for the given dataset: ", accuracy)

# Dataset:



**Output:**