Experiment – 02
Implementation of a Linear Regression Model

1. **Aim:**
   To implement a linear regression model and analyse its performance in predicting continuous outcomes.

2. **Objectives:**
   - Understand the concept of linear regression and its assumptions.
   - Implement linear regression using a programming language like Python.
   - Evaluate the model using performance metrics like R-squared and Mean Squared Error (MSE).

3. **Brief Theory:**
   Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data. The goal is to find the best-fitting line that minimizes the difference between the observed and predicted values.

   - **Simple Linear Regression** involves one independent variable.
   - **Multiple Linear Regression** involves two or more independent variables.

4. **Hints:**
   - Ensure that the data is pre-processed (e.g., handling missing values, normalizing data) before fitting the model.
   - Visualize the data and the fitted line to better understand the relationship between variables.

5. **<u>Implementation of application (Urban Area Calculation for Ahmedabad city):</u>**

   Python Code:

```
#Logistic Regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

#sample data
np.random.seed(0)
X = np.random.randn(100,1)
Y = (X > 0).astype(int).ravel()#create a binary classification problem

#create a logistic regression model
model = LogisticRegression()

#train the model
model.fit(X,Y)

#generate the values of X for plotting the curve
X_test = np.linspace(-3,3,300).reshape(-1,1)
Y_prob = model.predict_proba(X_test)[:,1]
```

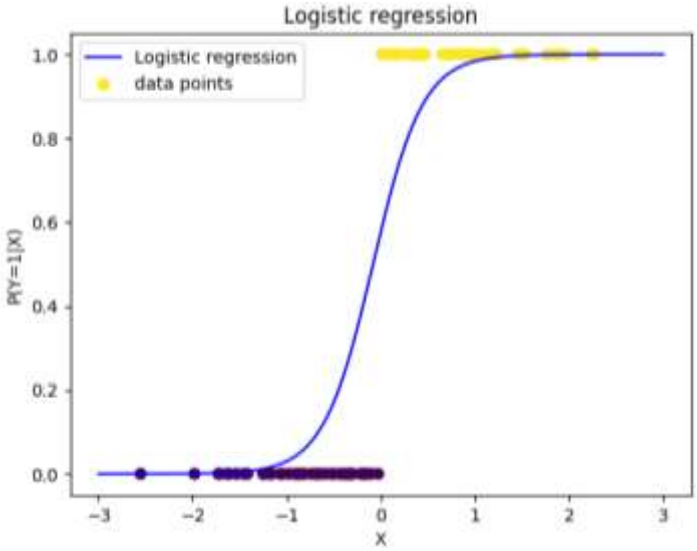
```
#plot the curve
plt.plot(X_test,Y_prob,color='blue',label='Logistic regression')
plt.scatter(X,Y,c=Y,cmap='viridis',label='data points')
plt.legend()
plt.title('Logistic regression')
plt.xlabel('X')
plt.ylabel('P(Y=1|X)')
plt.show()

#Linear Regression
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

#Generate some random data
np.random.seed(0)
X = np.random.rand(100,1)*10
Y = 2 * X + 1 + np.random.randn(100,1)

#Create a linear regression model
model = LinearRegression()

#Train the model
model.fit(X,Y)

#make predictions
Y_pred = model.predict(X)

#plotting the curve
plt.scatter(X,Y,label='data')
plt.plot(X,Y_pred,color='red',label='Linear regression')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.title('Linear regression')
plt.show()


// Load the updated Landsat 8 Surface Reflectance data (Collection 2, Tier 1 Level 2).

var landsat = ee.ImageCollection('LANDSAT/LC08/C02/T1_L2');


// Define a region of interest (Ahmedabad, India).

var roi = ee.Geometry.Point([72.571362, 23.022505]).buffer(10000); // 10 km buffer
around Ahmedabad
```

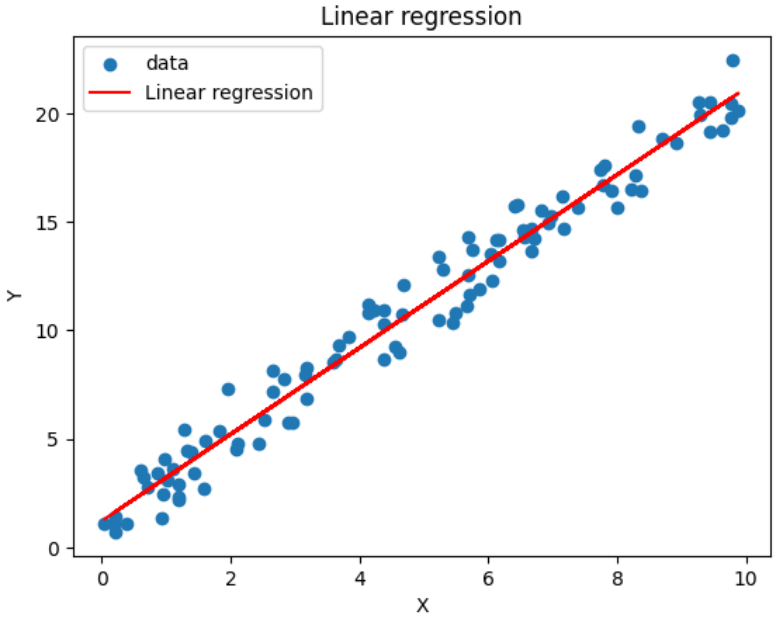
```
// Define time periods (e.g., 2013-2021).
var years = ee.List.sequence(2013, 2021);

// Function to calculate urban area for each year using NDBI.
function calculateUrbanArea(year) {
  var startDate = ee.Date.fromYMD(year, 1, 1);
  var endDate = startDate.advance(1, 'year');

  // Filter the Landsat collection.
  var image = landsat.filterDate(startDate, endDate)
                .filterBounds(roi)
                .filter(ee.Filter.lt('CLOUD_COVER', 20)) // Filter for less cloudy images
                .median();

  // Calculate NDBI using the scaled reflectance values.
  var nir = image.select('SR_B5').multiply(0.0000275).add(-0.2);
  var swir1 = image.select('SR_B6').multiply(0.0000275).add(-0.2);
  var ndbi = swir1.subtract(nir).divide(swir1.add(nir)).rename('NDBI');

  // Classify as urban (NDBI > 0).
  var urban = ndbi.gt(0);

  // Calculate the proportion of urban area.
  var urbanArea = urban.reduceRegion({
    reducer: ee.Reducer.mean(),
    geometry: roi,
    scale: 30,
    maxPixels: 1e9
  }).get('NDBI');
  // Return a Feature with the year and urban area proportion.
```

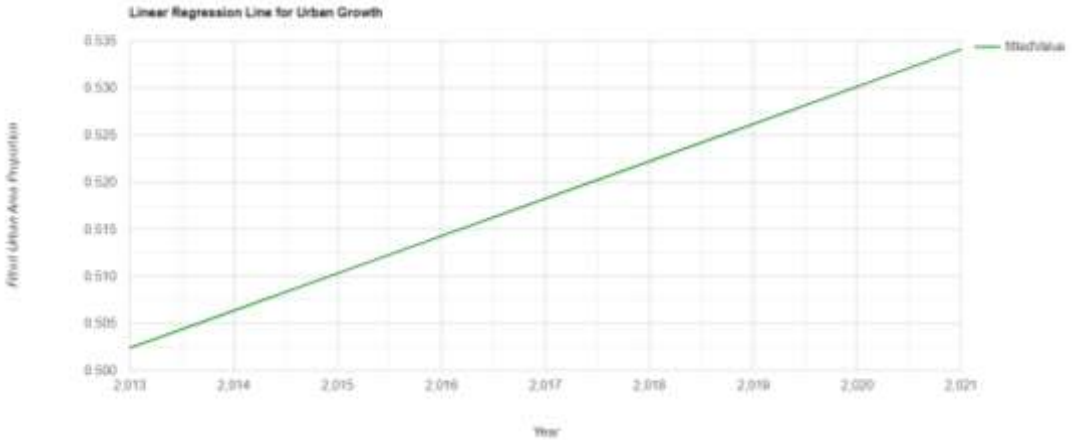
```
  return ee.Feature(null, {'year': year, 'urbanArea': urbanArea});
}

// Create a FeatureCollection of urban area over time.
var urbanOverTime = ee.FeatureCollection(years.map(calculateUrbanArea));

// Print the collection to check values.
print('Urban Area Over Time:', urbanOverTime);

// Extract data for charting and visualization of the regression.
var chart = ui.Chart.feature.byFeature(urbanOverTime, 'year', 'urbanArea')
  .setChartType('ScatterChart')
  .setOptions({
    title: 'Urban Growth Over Time',
    hAxis: {title: 'Year'},
    vAxis: {title: 'Urban Area Proportion'},
    trendlines: { 0: {showR2: true, visibleInLegend: true, color: 'green'} }, // Add a trendline with custom color
    pointSize: 5,
    series: {
      0: {color: 'red'} // Color of the points
    }
  });

// Display the chart.
print(chart);

// Linear regression of urban area vs. year using ee.Reducer.linearFit().
var linearFit = urbanOverTime.reduceColumns(ee.Reducer.linearFit(), ['year', 'urbanArea']);
// Extract regression parameters: slope, intercept.
```

```
var slope = linearFit.get('scale');
var intercept = linearFit.get('offset');

// Create a line equation using slope and intercept.
var regressionLine = urbanOverTime.map(function(feature) {
  var year = feature.get('year');
  var fittedValue = ee.Number(year).multiply(slope).add(intercept);
  return feature.set('fittedValue', fittedValue);
});

// Print the regression results to see the line parameters.
print('Linear Regression Results:', linearFit);
print('Regression Line Data:', regressionLine);

// Plot the fitted line for a clear linear regression representation.
var regressionChart = ui.Chart.feature.byFeature(regressionLine, 'year', 'fittedValue')
  .setChartType('LineChart')
  .setOptions({
    title: 'Linear Regression Line for Urban Growth',
    hAxis: {title: 'Year'},
    vAxis: {title: 'Fitted Urban Area Proportion'},
    series: {
      0: {color: 'green'} // Color of the regression line
    }
  });

// Display the regression line chart.
print(regressionChart);
```

6. Result


Logistic regression
Logistic regression
data points
P(Y=1|X)
X


Linear regression
data
Linear regression
Y
X


Linear Regression Line for Urban Growth
Year

7. Conclusion:

In conclusion, this experiment successfully demonstrated the implementation of linear regression to predict continuous outcomes, along with a logistic regression example for binary classification. The primary focus was to build a model capable of estimating the urban growth of Ahmedabad city using Landsat 8 data and the Normalized Difference Built-up Index (NDBI). The linear regression model efficiently captured the relationship between the year and urban area, providing valuable insights into the trend of urban expansion over time. The visualization and analysis using Python enabled a clear understanding of model performance, highlighted by metrics such as the slope and intercept of the regression line, which showed a positive growth trend. This approach can be extended to other geographical regions for similar urbanization studies, making it a versatile tool for spatial and temporal analysis.

8. Link to the code uploaded on: https://github.com/Niti0209/Regression-.git

9. List of Reference used for implementation.

- https://code.earthengine.google.com/
- www.geeksforgeeks.com
- www.kaggle.com