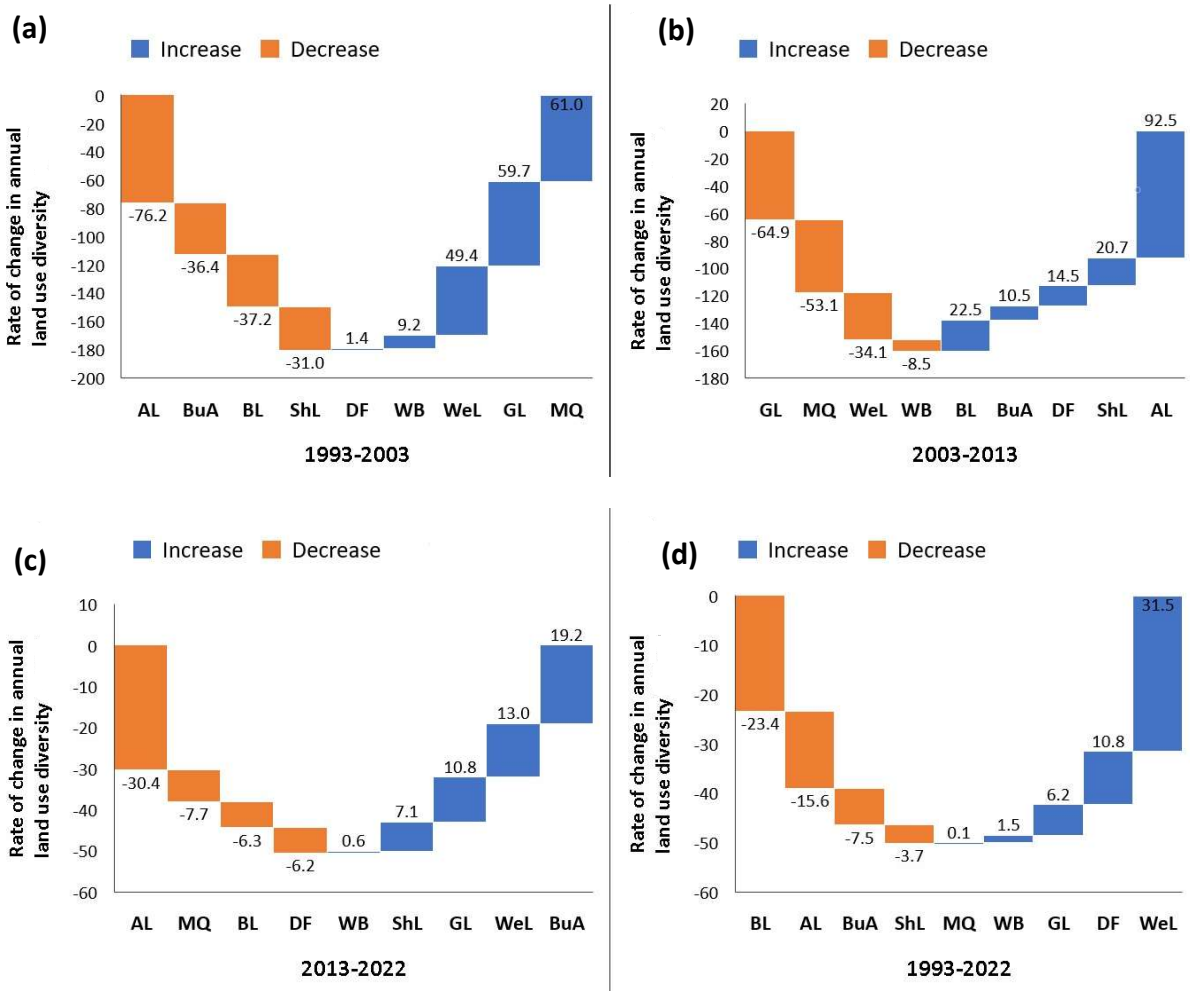



**Figure S1** a-d. Annual rate of land use diversity transfer-out (loss) & in -analysis (gain) in km² for EKU municipality from 1993-2022.

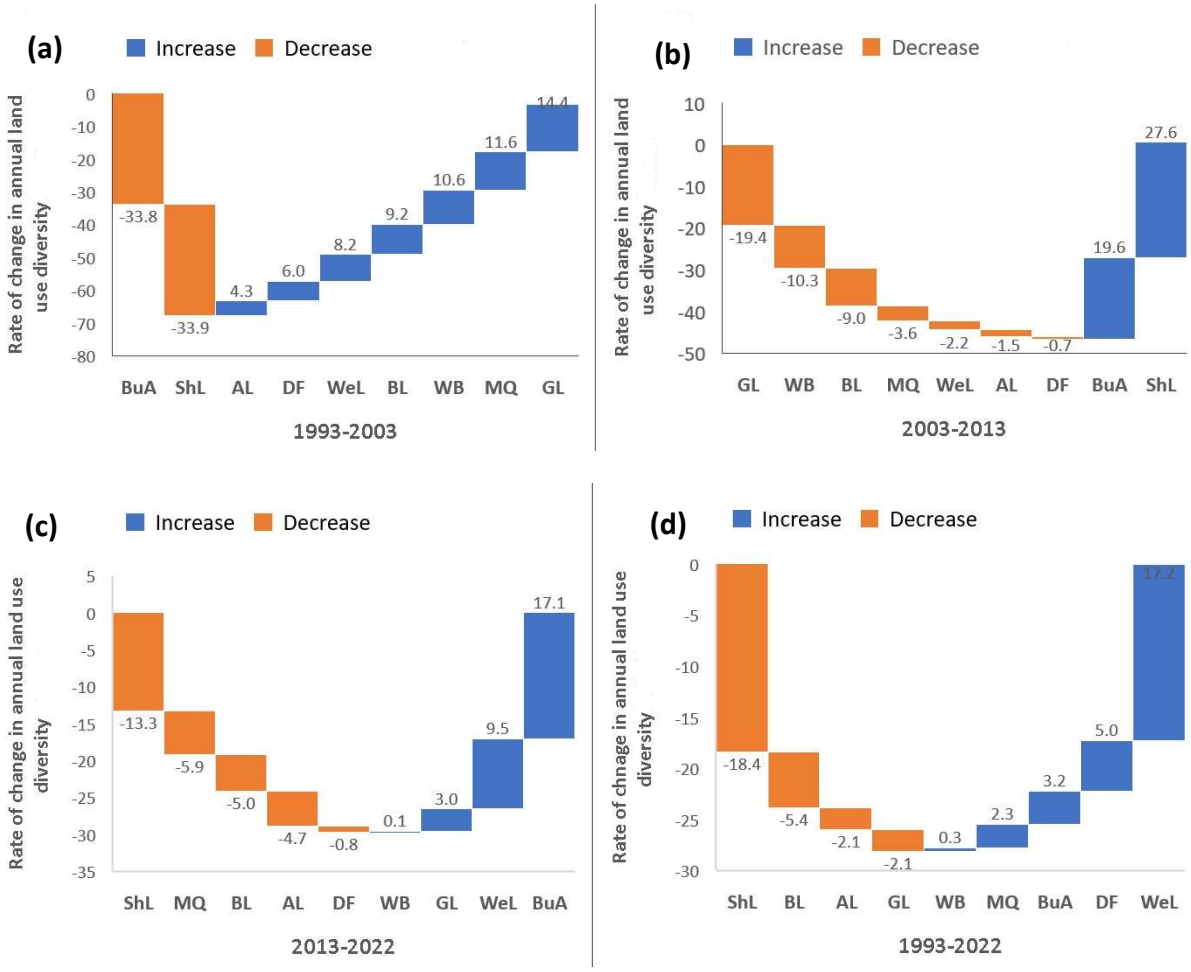



**Figure S2 a-d**. Annual rate of land use diversity transfer-out (loss) & in -analysis (gain) in km$^2$ for JHB metropolitan city from 1993-2022.

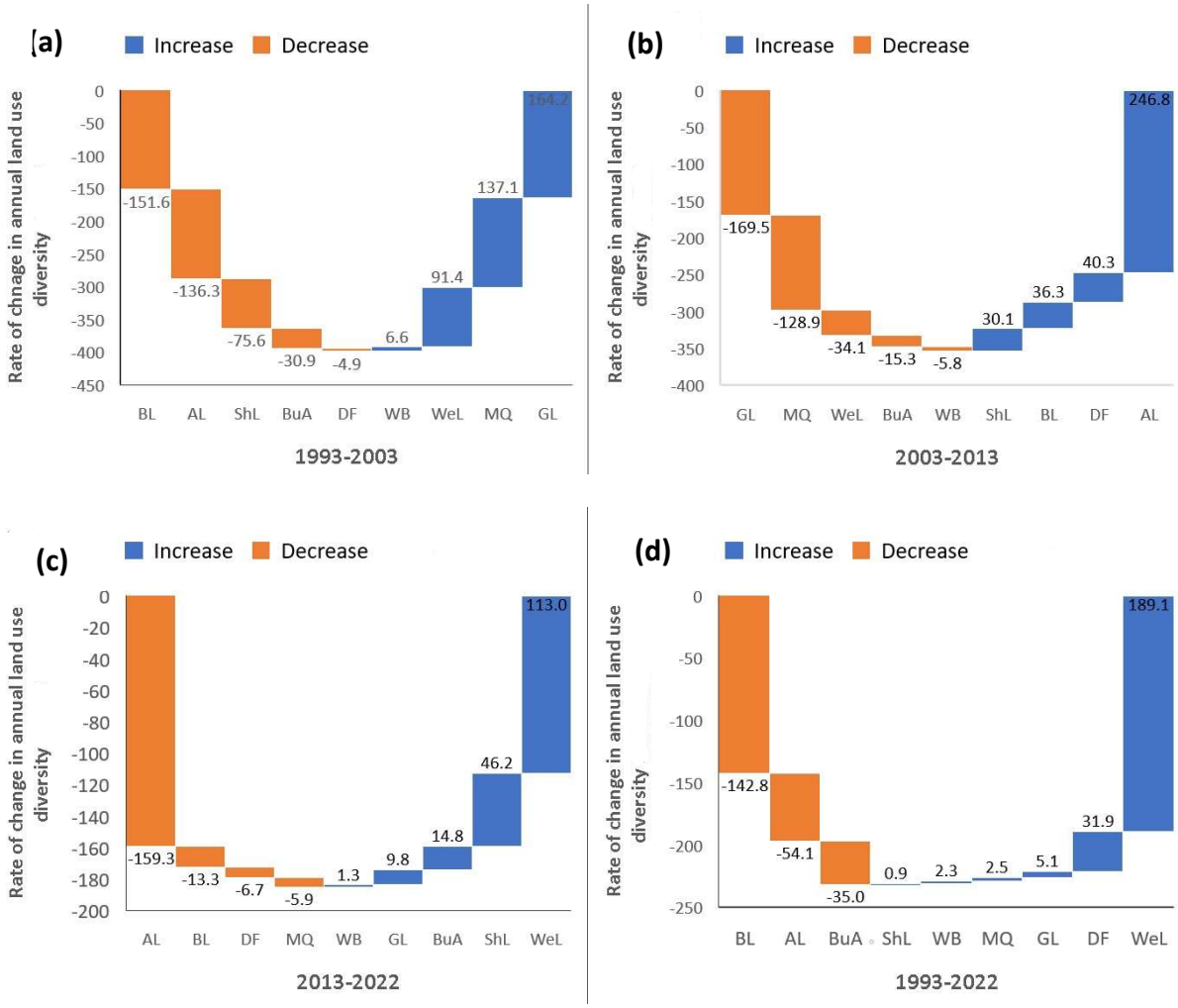



**Figure S3 a-d.** Annual rate of land use diversity transfer-out (loss) & in -analysis (gain) in km$^2$ for TSH metropolitan from 1993-2022.

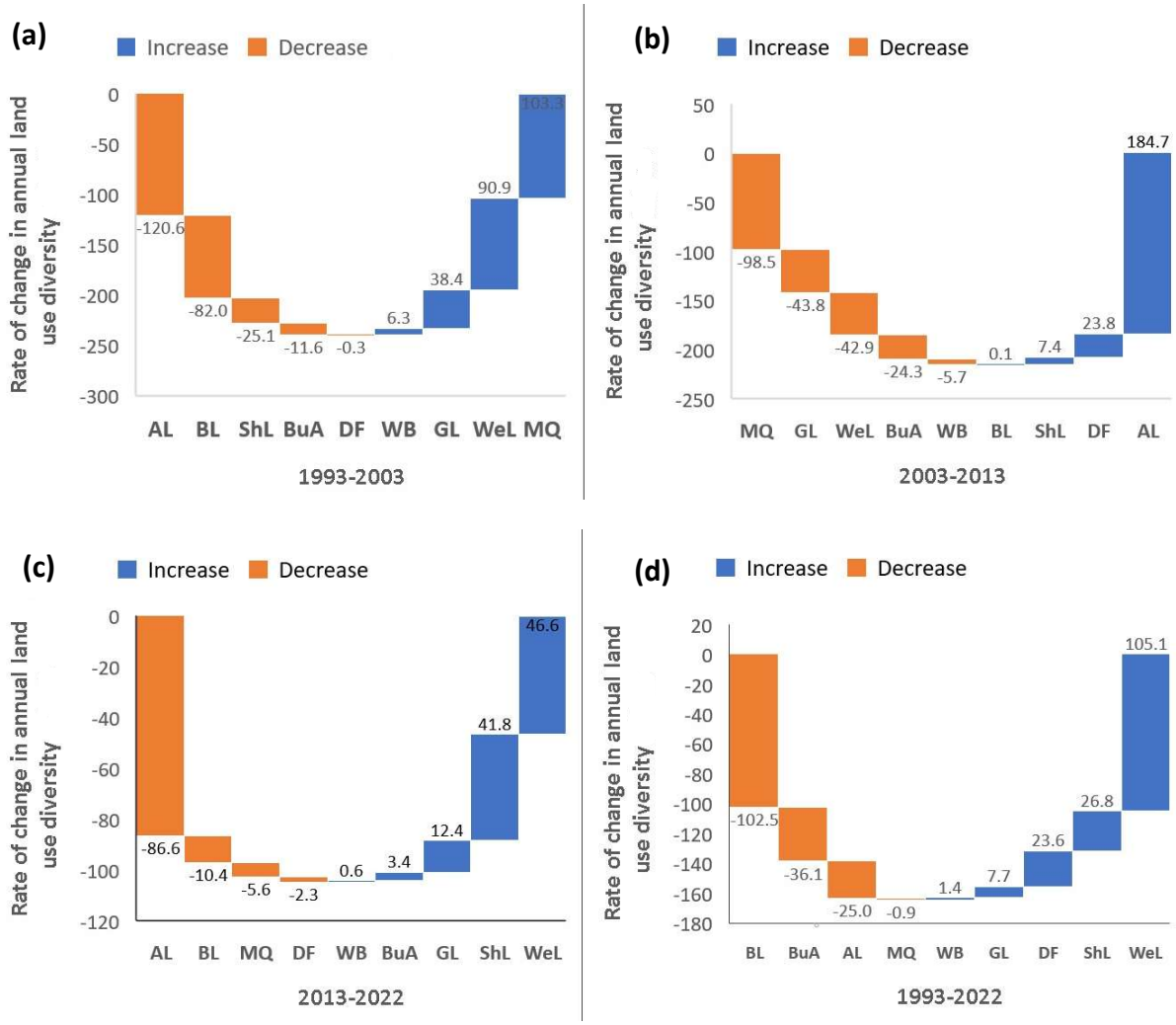



**Figure S4** a-d. Annual rate of land use diversity transfer-out (loss) & in -analysis (gain) in km² for DC48 municipality from 1993-2022.

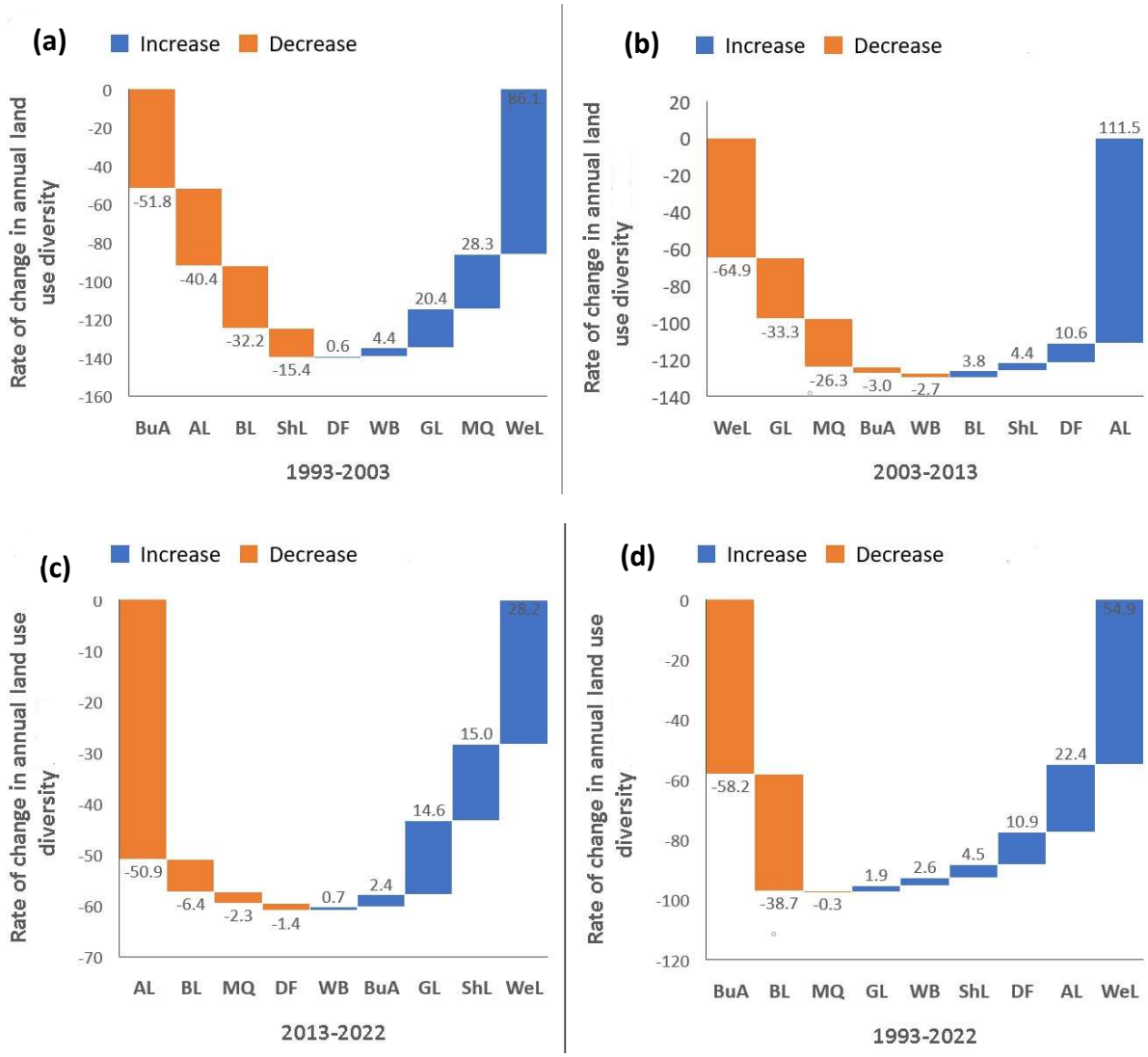
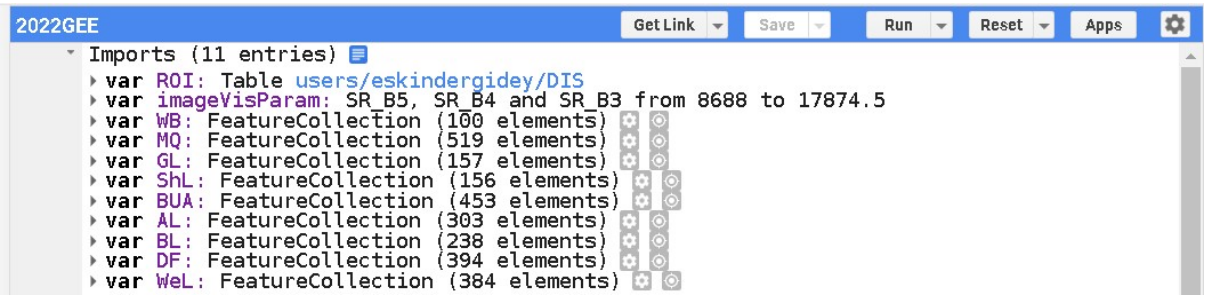


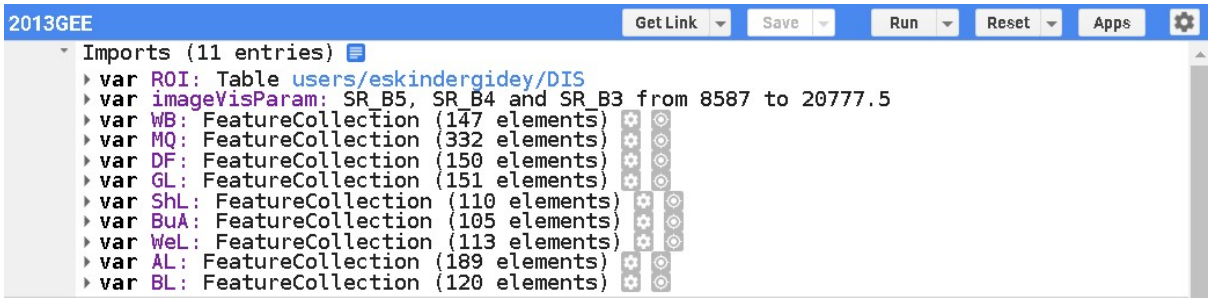

**Figure S5 a-d.** Annual rate of land use diversity transfer-out (loss) & in -analysis (gain) in km² for DC42 municipality from 1993-2022.

**RF-GEE scripts for land use diversity analysis**



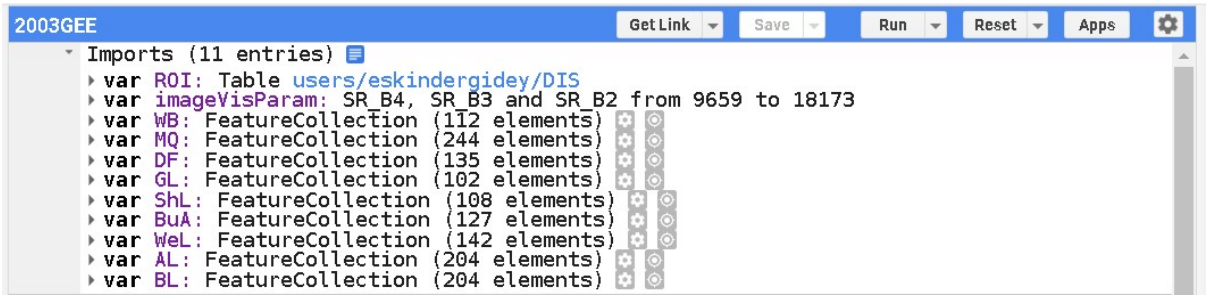

```
var Landsat8 = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")
var image = Landsat8.filterBounds(ROI)
        .filterDate('2022-08-01', '2022-08-26')
        .select('SR_B7','SR_B6','SR_B5','SR_B4','SR_B3','SR_B2', 'SR_B1')
        .filterMetadata('CLOUD_COVER', 'less_than', 1)
        .median()
        .clip(ROI)
//print(image.size());
print(image)
Map.addLayer(image, imageVisParam, 'True color')
Map.centerObject(ROI, 8)

////////////////// For selecting land use classes and merging training datasets (samples)
Points/////////////////////////////////////////
var training =
WB.merge(MQ).merge(GL).merge(ShL).merge(BUA).merge(AL).merge(BL).merge(DF).merge(WeL)//.
merge(R)
print(training)
///////////////// For sample points analysis ////////////////////////////////
var bands = ['SR_B7','SR_B6','SR_B5','SR_B4','SR_B3','SR_B2', 'SR_B1']
var input = image.select(bands)
var trainImage = input.sampleRegions({
  collection: training,
  properties: ['Class'],
  scale: 30
})
print(trainImage)

var trainData = trainImage.randomColumn()
var trainSet = trainData.filter(ee.Filter.lessThan('random', 0.7))
var testSet = trainData.filter(ee.Filter.greaterThanOrEquals('random', 0.7))
////////////// RF Classification Model ////////////////////////////////////////////
var classifier = ee.Classifier.smileRandomForest(10).train({
  features: trainSet,
  classProperty: 'Class',
  inputProperties: bands})

var classified = input.classify(classifier)
Map.addLayer(classified, {min: 0, max: 9, palette: ['#0b32d6', '#ff581f', '#0b4a8b', '#ffc82d', '#00ffff',
'#bf04c2', '#ff0000', '#00ff00', '#6dffe7', '#999900']}, 'Classification')
```

```
//////////////////////////// Accuracy assessment
//////////////////////////////////////////////////////
//print(classifier.confusionMatrix());

var confusionMatrix = ee.ConfusionMatrix(testSet.classify(classifier)
.errorMatrix({
  actual: 'Class',
  predicted: 'classification'}))
print ('confusionMatrix:', confusionMatrix);
print ('Overall Accuracy:', confusionMatrix.accuracy());

//////////////////////////// To calculate kappa statistics ////////////////////////////////////////////
print("Kappa Statistics", confusionMatrix.kappa());

//////////////////////// To calculate consumer's accuracy, also known user's accuracy
////////////////////
print("consumer's accuracy", confusionMatrix.consumersAccuracy());

//////////////////////// To calculate producer's accuracy, also known user's accuracy
////////////////////
print("producer's accuracy", confusionMatrix.producersAccuracy());

//////////////////////////// To calculate area ////////////////////////////////////////////
var area = ee.Image.pixelArea().divide(1e6).addBands(classified).reduceRegion({
  reducer: ee.Reducer.sum().group({
    groupField: 1,
    groupName: "classified"}),
    geometry: ROI,
    scale: 1000,
    maxPixels: 1e13})
print(area)

//////////////////////// To export results into google drive
////////////////////////////////////////////////
Export.image.toDrive({
  image: classified,
  description: "Rf-2022b2",
  fileFormat: 'GeoTIFF',
  scale: 30,
  region: ROI,
  })
```



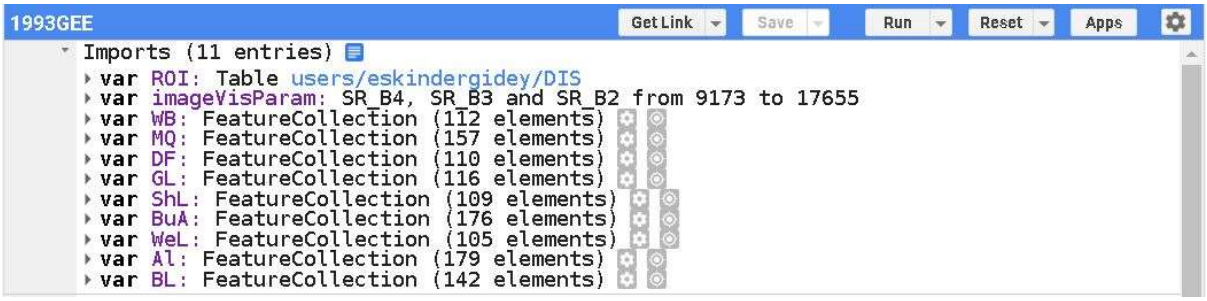

```
var Landsat8 = ee.ImageCollection("LANDSAT/LC08/C02/T1_L2")
var image = Landsat8.filterBounds(ROI)
        .filterDate('2013-07-24', '2013-08-16')
        .select('SR_B7','SR_B6','SR_B5','SR_B4','SR_B3','SR_B2', 'SR_B1')
        .filterMetadata('CLOUD_COVER', 'less_than', 1)
        .median()
        .clip(ROI)
//print(image.size());
print(image)
Map.addLayer(image, imageVisParam, 'True color')
Map.centerObject(ROI, 8)

////////////////// For selecting land use classes and merging training datasets (samples)
Points//////////////////////////////////////////
var training =
WB.merge(MQ).merge(DF).merge(GL).merge(ShL).merge(BuA).merge(WeL).merge(AL).merge(BL)//.
merge(R)
print(training)
///////////////// To create sample points ////////////////////////////////
var bands = ['SR_B7','SR_B6','SR_B5','SR_B4','SR_B3','SR_B2', 'SR_B1']
var input = image.select(bands)
var trainImage = input.sampleRegions({
  collection: training,
  properties: ['Class'],
  scale: 30
})
print(trainImage)

var trainData = trainImage.randomColumn()
var trainSet = trainData.filter(ee.Filter.lessThan('random', 0.7))
var testSet = trainData.filter(ee.Filter.greaterThanOrEquals('random', 0.7))
/////////////// Classification Model ////////////////////////////////////////////
var classifier = ee.Classifier.smileRandomForest(10).train({
  features: trainSet,
  classProperty: 'Class',
  inputProperties: bands})

var classified = input.classify(classifier)
Map.addLayer(classified, {min: 0, max: 9, palette: ['#0b32d6', '#ff581f', '#0b4a8b', '#ffc82d', '#00ffff',
'#bf04c2', '#ff0000', '#00ff00', '#6dffe7', '#999900']}, 'Classification')

////////////////////////// Accuracy assessment
//////////////////////////////////////////////////////
```

```
//print(classifier.confusionMatrix());

var confusionMatrix = ee.ConfusionMatrix(testSet.classify(classifier)
.errorMatrix({
  actual: 'Class',
  predicted: 'classification'}))
print ('confusionMatrix:', confusionMatrix);
print ('Overall Accuracy:', confusionMatrix.accuracy());

////////////////////////// To calculate kappa statistics ////////////////////////////////////////////
print("Kappa Statistics", confusionMatrix.kappa());

////////////////////////// To calculate consumer's accuracy, also known user's accuracy
//////////////////
print("consumer's accuracy", confusionMatrix.consumersAccuracy());

//////////////////////////  To calculate producer's accuracy, also known user's accuracy
//////////////////
print("producer's accuracy", confusionMatrix.producersAccuracy());

////////////////////////////// To calculate area ////////////////////////////////////////////////
var area = ee.Image.pixelArea().divide(1e6).addBands(classified).reduceRegion({
  reducer: ee.Reducer.sum().group({
    groupField: 1,
    groupName: "classified"}),
    geometry: ROI,
    scale: 1000,
    maxPixels: 1e13})
print(area)

////////////////////////// To export results into google drive
////////////////////////////////////////////////
Export.image.toDrive({
  image: classified,
  description: "Rf-2013b2",
  fileFormat: 'GeoTIFF',
  scale: 30,
  region: ROI,
  })
```




```
var Landsat5 = ee.ImageCollection("LANDSAT/LT05/C02/T1_L2")
var image = Landsat5.filterBounds(ROI)
        .filterDate('2003-03-01', '2003-12-30')
        .select('SR_B7','SR_B5','SR_B4','SR_B3','SR_B2','SR_B1')
        .filterMetadata('CLOUD_COVER', 'less_than', 9)
        .median()
        .clip(ROI)
//print(image.size());
print(image)
Map.addLayer(image, imageVisParam, 'True color')
Map.centerObject(ROI, 8)


////////////////// For selecting land use classes and merging training datasets (samples)
Points//////////////////////////////////////////
var training =
WB.merge(MQ).merge(DF).merge(GL).merge(ShL).merge(BuA).merge(WeL).merge(AL).merge(BL)//.
merge(R)
print(training)
///////////////// For sample points analysis ////////////////////////////////
////////////////////////////////
var bands = ['SR_B7','SR_B5','SR_B4','SR_B3','SR_B2', 'SR_B1']
var input = image.select(bands)
var trainImage = input.sampleRegions({
  collection: training,
  properties: ['Class'],
  scale: 30
})
print(trainImage)

var trainData = trainImage.randomColumn()
var trainSet = trainData.filter(ee.Filter.lessThan('random', 0.7))
var testSet = trainData.filter(ee.Filter.greaterThanOrEquals('random', 0.7))
/////////////// Classification Model ////////////////////////////////////////////
var classifier = ee.Classifier.smileRandomForest(10).train({
  features: trainSet,
  classProperty: 'Class',
  inputProperties: bands})

var classified = input.classify(classifier)
Map.addLayer(classified, {min: 0, max: 9, palette: ['#0b32d6', '#ff581f', '#0b4a8b', '#ffc82d', '#00ffff',
'#bf04c2', '#ff0000', '#00ff00', '#6dffe7', '#999900']}, 'Classification')
```

```
//////////////////////////// For accuracy assessment
/////////////////////////////////////////////////////
//print(classifier.confusionMatrix());

var confusionMatrix = ee.ConfusionMatrix(testSet.classify(classifier)
.errorMatrix({
  actual: 'Class',
  predicted: 'classification'}))
print ('confusionMatrix:', confusionMatrix);
print ('Overall Accuracy:', confusionMatrix.accuracy());

//////////////////////// Calculate Kappa Statistics ////////////////////////////////////////////
print("Kappa Statistics", confusionMatrix.kappa());

//////////////////////// To calculate consumer's accuracy, also known user's accuracy
//////////////////
print("consumer's accuracy", confusionMatrix.consumersAccuracy());

////////////////////////  To calculate producer's accuracy, also known user's accuracy
//////////////////
print("producer's accuracy", confusionMatrix.producersAccuracy());

//////////////////////////// To calculate area ////////////////////////////////////////////
var area = ee.Image.pixelArea().divide(1e6).addBands(classified).reduceRegion({
  reducer: ee.Reducer.sum().group({
    groupField: 1,
    groupName: "classified"}),
    geometry: ROI,
    scale: 1000,
    maxPixels: 1e13})
print(area)

//////////////////////// To export results into google drive
/////////////////////////////////////////////////
Export.image.toDrive({
  image: classified,
  description: "RF-2003b",
  fileFormat: 'GeoTIFF',
  scale: 30,
  region: ROI,
  })
```




```
var Landsat5 = ee.ImageCollection("LANDSAT/LT05/C02/T1_L2")
var image = Landsat5.filterBounds(ROI)
        .filterDate('1993-08-02', '1993-08-30')
        .select('SR_B7', 'SR_B5', 'SR_B4', 'SR_B3', 'SR_B2', 'SR_B1')
        .filterMetadata('CLOUD_COVER', 'less_than', 1)
        .median()
        .clip(ROI)
//print(image.size());
print(image)
Map.addLayer(image, imageVisParam, 'True color')
Map.centerObject(ROI, 8)

////////////////// For selecting land use classes and merging training datasets (samples)
Points/////////////////////////////////////////
var training =
WB.merge(MQ).merge(DF).merge(GL).merge(ShL).merge(BuA).merge(WeL).merge(Al).merge(BL)//.
merge(R)
print(training)
////////////////// For creating training data sets ////////////////////////////////
var bands = ['SR_B7','SR_B5','SR_B4','SR_B3','SR_B2', 'SR_B1']
var input = image.select(bands)
var trainImage = input.sampleRegions({
  collection: training,
  properties: ['Class'],
  scale: 30
})
print(trainImage)

var trainData = trainImage.randomColumn()
var trainSet = trainData.filter(ee.Filter.lessThan('random', 0.7))
var testSet = trainData.filter(ee.Filter.greaterThanOrEquals('random', 0.7))
/////////////// RF Classification Model ////////////////////////////////////////////
var classifier = ee.Classifier.smileRandomForest(10).train({
  features: trainSet,
  classProperty: 'Class',
  inputProperties: bands})

var classified = input.classify(classifier)
Map.addLayer(classified, {min: 0, max: 9, palette: ['#0b32d6', '#ff581f', '#0b4a8b', '#ffc82d', '#00ffff',
'#bf04c2', '#ff0000', '#00ff00', '#6dffe7', '#999900']}, 'Classification')

//////////////////////////// Accuracy assessment
////////////////////////////////////////////////////////
```

```
//print(classifier.confusionMatrix());

var confusionMatrix = ee.ConfusionMatrix(testSet.classify(classifier)
.errorMatrix({
  actual: 'Class',
  predicted: 'classification'}))
print ('confusionMatrix:', confusionMatrix);
print ('Overall Accuracy:', confusionMatrix.accuracy());

////////////////////////// To calculate kappa statistics ////////////////////////////////////////////
print("Kappa Statistics", confusionMatrix.kappa());

////////////////////////// To calculate consumer's accuracy, also known user's accuracy
//////////////////
print("consumer's accuracy", confusionMatrix.consumersAccuracy());

////////////////////////// To calculate producer's accuracy, also known user's accuracy
//////////////////
print("producer's accuracy", confusionMatrix.producersAccuracy());

///////////////////////////// To calculate area ////////////////////////////////////////////////
var area = ee.Image.pixelArea().divide(1e6).addBands(classified).reduceRegion({
  reducer: ee.Reducer.sum().group({
    groupField: 1,
    groupName: "classified"}),
    geometry: ROI,
    scale: 1000,
    maxPixels: 1e13})
print(area)

////////////////////////// To export results into google drive
////////////////////////////////////////////////
Export.image.toDrive({
  image: classified,
  description: "RF-1993b",
  fileFormat: 'GeoTIFF',
  scale: 30,
  region: ROI,
  })
```