Kerala Flood Data Preprocessing

```
#Importing libraries

import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

#Read CSV File

df=pd.read_csv('kerala_flood.csv')
df
```

```
    SUBDIVISION  YEAR   JAN   FEB  MARCH  APRIL    MAY    JUNE    JULY
AUG  \
0        KERALA  1901  28.7  44.7   51.6  160.0  174.7   824.6   743.0
357.5
1        KERALA  1902   6.7   2.6   57.3   83.9  134.5   390.9  1205.0
315.8
2        KERALA  1903   3.2  18.6    3.1   83.6  249.7   558.6  1022.5
420.2
3        KERALA  1904  23.7   3.0   32.2   71.5  235.7  1098.2   725.5
351.8
4        KERALA  1905   1.2  22.3    9.4  105.9  263.3   850.2   520.5
293.6
..          ...   ...   ...   ...    ...    ...    ...     ...     ...
...
110      KERALA  2011  20.5  45.7   24.1  165.2  124.2   788.5   536.8
492.7
111      KERALA  2012   7.4  11.0   21.0  171.1   95.3   430.3   362.6
501.6
112      KERALA  2013   3.9  40.1   49.9   49.3  119.3  1042.7   830.2
369.7
113      KERALA  2014   4.6  10.3   17.9   95.7  251.0   454.4   677.8
733.9
114      KERALA  2015   3.1   5.8   50.1  214.1  201.8   563.6   406.0
252.2

     ...    NOV    DEC  ANNUAL  JAN-FEB  MARCH-MAY  JUNE_SEPT  OCT_DEC
FLOOD  \
0    ...  350.8   48.4  3248.6     73.4      386.2     2122.8    666.1
0
1    ...  158.3  121.5  3326.6      9.3      275.7     2403.4    638.2
1
2    ...  157.0   59.0  3271.2     21.7      336.3     2343.0    570.1
0
3    ...   33.9    3.3  3129.7     26.7      339.4     2398.2    365.3
0
```

```
4    ...   74.4    0.2  2741.6     23.4      378.5     1881.5    458.1
0
..   ...    ...    ...     ...      ...        ...        ...      ...
...
110  ...  169.7   49.5  3035.1     66.2      313.5     2209.1    446.3
0
111  ...  112.9    9.4  2151.1     18.3      287.4     1535.6    309.8
0
112  ...  154.9   17.0  3255.4     43.9      218.5     2561.2    431.8
1
113  ...   99.5   47.2  3046.4     14.9      364.5     2164.8    502.1
0
114  ...  223.6   79.4  2600.6      8.9      465.9     1514.7    611.1
0

        AVGJUNE    SUB
0    274.866667  649.9
1    130.300000  256.4
2    186.200000  308.9
3    366.066667  862.5
4    283.400000  586.9
..          ...    ...
110  262.833333  664.3
111  143.433333  335.0
112  347.566667  923.4
113  151.466667  203.4
114  187.866667  361.8

[115 rows x 22 columns]
```

```python
#create function to check for missing values in the data

def count_of_null(df):
    count=df.isnull().sum().sum()
    return count

# print missing value count in the data
count_null = count_of_null(df)
print(count_null)
```

```
0
```

```python
#create function to check duplicate values in the data
def check_duplicates(df):
    count=df.duplicated().sum().sum()
    return count

# print duplicate values in the data
count_duplicates = check_duplicates(df)
print(count_duplicates)
```

```
0
```

```python
#create a function to check information about the data
def about_data (df):
        about=df.info()
        return about

#print information about the data
info=about_data (df)
print(info)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 115 entries, 0 to 114
Data columns (total 22 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   SUBDIVISION  115 non-null    object 
 1   YEAR         115 non-null    int64  
 2   JAN          115 non-null    float64
 3   FEB          115 non-null    float64
 4   MARCH        115 non-null    float64
 5   APRIL        115 non-null    float64
 6   MAY          115 non-null    float64
 7   JUNE         115 non-null    float64
 8   JULY         115 non-null    float64
 9   AUG          115 non-null    float64
 10  SEPT         115 non-null    float64
 11  OCT          115 non-null    float64
 12  NOV          115 non-null    float64
 13  DEC          115 non-null    float64
 14  ANNUAL       115 non-null    float64
 15  JAN-FEB      115 non-null    float64
 16  MARCH-MAY    115 non-null    float64
 17  JUNE_SEPT    115 non-null    float64
 18  OCT_DEC      115 non-null    float64
 19  FLOOD        115 non-null    int64  
 20  AVGJUNE      115 non-null    float64
 21  SUB          115 non-null    float64
dtypes: float64(19), int64(2), object(1)
memory usage: 19.9+ KB
None
```

```python
# create function to print unique values in the dataframe

def print_unique_values(df):
    for column in df.columns:
        unique_values = df[column].unique()
        print(f"Column '{column}' has {len(unique_values)} unique 
values:")
        print(unique_values)
print_unique_values(df)
```

```
Column 'SUBDIVISION' has 1 unique values:
['KERALA']
Column 'YEAR' has 115 unique values:
[1901 1902 1903 1904 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914
 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928
 1929 1930 1931 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942
 1943 1944 1945 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956
 1957 1958 1959 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970
 1971 1972 1973 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984
 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998
 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012
 2013 2014 2015]
Column 'JAN' has 86 unique values:
[28.7  6.7  3.2 23.7  1.2 26.7 18.8  8.  54.1  2.7  3.   1.9  3.1  0.7
 16.9  0.   2.9 42.9 43.  35.2 30.5 24.7 19.3  4.1 28.6 12.7 12.8 10.8
  3.3  0.1  1.  74.5 23.9  6.5  0.3 13.6  0.6 15.9  2.4 83.5  6.4  4.4
  1.8 22.2  6.6  5.2 13.1 23.5  4.2  7.9  9.4 13.7 28.4 30.2  1.1  9.1
 14.3  7.3 12.9 31.6  2.6  1.6  7.   0.2 36.8 61.2  5.6  0.8 10.3 14.9
 10.9 24.3  2.8  2.1  6.  11.7 16.5  4.7 19.8  8.1  0.5 18.6 20.5  7.4
  3.9  4.6]
Column 'FEB' has 94 unique values:
[44.7  2.6 18.6  3.  22.3  7.4  4.8 20.8 11.8 25.7  4.3 15.   5.2  6.8
 23.5  7.8 47.6  5.   6.1  5.5  4.7 21.4  0.7  2.9 16.5  5.8 35.3 65.9
 29.8 10.8  0.3 19.3  9.3  1.7  8.3 21.2 79.   3.6  1.5  4.6 14.6 26.6
  9.9  5.4 27.3  1.8 53.7  6.5 48.2 22.6  2.8  6.3 11.7 16.   8.4 31.3
 54.7 24.8  0.9  6.9  0.1 30.5  6.4 17.6 18.5  7.5 15.7 14.7 30.  60.
 18.7  0.8 17.5  0.   4.4 17.8 27.1  9.1  2.1 23.8 57.8 28.3  8.7 50.9
  8.1  7.   0.5  5.6 30.3  1.  45.7 11.  40.1 10.3]
Column 'MARCH' has 105 unique values:
[5.160e+01 5.730e+01 3.100e+00 3.220e+01 9.400e+00 9.900e+00 5.570e+01
 3.820e+01 6.130e+01 2.330e+01 1.820e+01 1.120e+01 2.070e+01 1.810e+01
 4.270e+01 2.200e+01 7.940e+01 3.280e+01 3.390e+01 2.410e+01 1.500e+01
 1.630e+01 7.890e+01 6.660e+01 7.690e+01 2.310e+01 4.960e+01 5.130e+01
 5.890e+01 3.900e+01 1.920e+01 2.860e+01 3.690e+01 4.770e+01 1.160e+02
 5.870e+01 5.330e+01 2.490e+01 3.800e+00 1.270e+01 2.320e+01 3.840e+01
 6.160e+01 1.084e+02 9.800e+01 4.820e+01 4.500e+00 3.110e+01 4.160e+01
 2.080e+01 1.850e+01 9.060e+01 2.820e+01 1.510e+01 2.570e+01 6.300e+00
 4.440e+01 1.140e+01 3.960e+01 6.980e+01 6.720e+01 2.830e+01 6.770e+01
 2.460e+01 8.920e+01 1.940e+01 2.580e+01 2.000e+01 2.500e+00 1.230e+01
 1.600e+01 6.340e+01 2.100e+01 3.140e+01 1.170e+01 2.850e+01 2.190e+01
 9.000e-01 9.530e+01 2.930e+01 4.300e+00 3.810e+01 3.010e+01 1.800e+01
 3.320e+01 1.000e-01 2.010e+01 3.730e+01 1.440e+01 3.610e+01 8.100e+00
 2.140e+01 2.150e+01 7.000e+00 3.570e+01 8.210e+01 3.790e+01 2.530e+01
 9.070e+01 7.300e+00 2.172e+02 6.260e+01 4.990e+01 1.790e+01
5.010e+01]
Column 'APRIL' has 113 unique values:
[160.   83.9  83.6  71.5 105.9  59.4 170.8 102.9  93.8 124.5  51.
122.7
  75.7  32.7 106.   82.4  38.1  51.3  65.9 172.  171.3  89.6  43.5
111.
```

```
  93.4  55.8  86.5 121.1 210.7 102.7 126.9 113.  139.5  92.4 120.7
34.
 175.5 164.5 172.8 126.5 101.9 180.3 107.5  61.6 104.1 139.8 142.2
125.
  98.1  68.5 175.9 112.2 132.4 136.9 125.9 151.6  70.2 135.1 150.7
206.6
  94.1  95.1  96.3  83.3 109.8 167.4  70.1 133.3 117.2 132.7  87.5
131.5
 128.  123.8 134.5 102.3  73.9  42.  114.8  75.9  60.4  13.1 162.1
66.6
  63.1  57.2 177.6 141.5  41.8  97.   43.   66.5 154.5 134.9 124.3
60.6
  61.1 111.6 238.  117.3 134.4 113.2 205.9  65.3 138.5 108.4  69.
138.9
 165.2 171.1  49.3  95.7 214.1]
Column 'MAY' has 115 unique values:
[174.7 134.5 249.7 235.7 263.3 160.8 101.4 142.6 473.2 148.8 180.6
217.3
 198.8 164.2 154.5 199.  122.9 683.  247.   87.7 104.1 293.6  80.
185.4
 258.2 222.6 265.4  81.9 148.  404.9 131.7 646.5 738.8 106.7  56.6
466.5
 137.1 179.6 105.1 217.4 417.5 191.9 478.4 212.7  53.4  83.   85.7
212.3
 440.  242.  148.5 214.6  55.4 179.5 544.2 351.3 381.2 353.5 347.2
540.
 500.5 472.4 157.1  94.8 214.5  95.2 244.9  90.  227.4 289.1 317.5
436.
 119.9 221.5 162.2  75.8 306.4 396.8 127.7 105.3 166.3 148.2  76.
84.6
 254.2 126.7 108.3 157.2 169.4 488.5 113.4 218.4 159.  141.3 355.6
74.3
 133.6 151.6 453.2 124.5 238.6 330.8  91.  610.9 134.8 521.2 192.7
81.2
 191.6 190.6 124.2  95.3 119.3 251.  201.8]
Column 'JUNE' has 113 unique values:
[ 824.6  390.9  558.6 1098.2  850.2  414.9  770.9  592.6  704.7  680.
  990.   948.2  541.7  565.3  696.1  920.2  703.7  464.3  636.8  964.3
  489.1  663.1  722.5 1011.7  688.8  563.9  720.2  590.7  946.6  633.1
  341.   859.3  852.9  431.3  620.8  485.6  681.6  625.8  606.4  797.6
  813.6  794.5  498.9  549.8  919.   556.1  910.2  536.3  638.3  774.1
  576.7  340.5  798.3  782.4  755.4  872.   713.3  872.8  480.3 1005.2
  244.9  393.3  379.4  597.7  496.2  696.4  550.5  535.3  889.6  401.8
  617.   266.9  864.4  196.8  599.6  758.1  582.9  745.9  912.4  612.2
  322.8  842.6  828.7  597.9  572.6  511.3  657.5  528.6 1096.1  819.3
  657.1  845.   493.4  572.4  544.2  732.5  607.3  633.8  715.3  503.1
  566.7  673.4  619.2  482.4  705.9  469.9  438.2  667.5  788.5  430.3
 1042.7  454.4  563.6]
Column 'JULY' has 113 unique values:
[ 743.  1205.  1022.5  725.5  520.5  954.2  760.4  902.2  782.3  484.1
```

```
  705.3  833.6  763.2  857.7  775.6  513.9  342.7  167.5  648.   940.8
  639.8 1025.1 1008.7 1526.5  593.5  885.2  888.2  420.6  844.   401.7
  653.9  716.4  773.4  415.   687.3  672.1  970.5  648.6  749.6  877.3
  517.9  828.8  831.6  614.1  704.   671.7  669.3  619.   758.7  905.7
  544.6  430.  1027.6  640.5  392.8  466.8  835.3  622.7 1155.7  750.9
 1146.5  951.1  720.2  754.2  465.1  601.9  741.4 1308.9  818.8  558.1
  714.4  583.5 1004.2  531.3  641.5  753.3  686.7  662.2  754.   489.8
  511.5  583.2  653.6  388.9  324.8  221.   502.8  450.7  635.4  905.5
  767.8  776.1  955.5  702.5  696.   641.4  700.4  343.2  598.5  318.7
  532.   385.4  832.7  804.   966.3  505.1  924.9  629.   536.8  362.6
  830.2  677.8  406. ]
Column 'AUG' has 113 unique values:
[ 357.5  315.8  420.2  351.8  293.6  442.8  981.5  352.9  258.   473.8
  178.6  534.4  247.2  402.2  298.8  396.9  335.1  376.   484.2  235.
  641.9  320.6  943.   624.   554.1  536.   315.   553.2  293.9  273.4
 1199.2  423.2  479.5  337.2  280.9  367.9  281.2  287.9  459.9  610.8
  458.5  329.3  183.3  230.7  695.6  739.6  487.9  445.2  387.3  190.6
  413.6  356.4  467.   236.   319.5  358.8  526.6  397.3  336.8  678.3
  510.7  511.   548.   296.1  202.1  508.4  380.7  284.8  554.8  385.2
  294.9  487.5  533.6  675.9  342.6  234.2  516.8  383.7  438.1  495.6
  495.   579.9  284.4  315.3  340.3  396.6  379.8  285.5  370.8  465.5
  508.   301.9  479.9  457.3  327.4  371.8  266.3  566.5  361.3  438.2
  350.3  417.9  291.   432.6  489.6  349.   269.3  356.   492.7  501.6
  369.7  733.9  252.2]
Column 'SEPT' has 114 unique values:
[197.7 491.6 341.8 222.7 217.2 131.2 225.  175.9 195.4 248.6  60.2
136.8
 176.9 241.  396.6 339.3 470.3  96.4 255.9 178.  156.7 222.4 254.3
289.1
 158.8 322.7 335.6  75.9 268.9 411.5 163.2 317.3 469.7  48.4 283.3
286.7
 139.8 223.2 134.1  68.2 257.9  99.8 257.6 155.  110.9 199.4 394.5
166.6
 354.5 411.6 313.8  57.4 100.5 201.6 438.5 178.4  41.3  86.1 405.5
371.2
 399.3 394.9 223.9 398.2 150.1 293.2 145.8 325.4 216.4 212.5 331.2
185.7
  61.3 383.6 457.7 116.2 201.3 119.4 211.7 139.5 376.6  70.6 421.1
171.1
 117.6 235.4 157.  451.7 271.1 103.3  48.5 297.5  88.  212.6 280.
342.7
 292.2 517.6 195.8 216.8  99.   93.6 192.8 414.7 474.8 526.7 347.
326.5
 275.6 391.2 241.1 318.6 298.8 292.9]
Column 'OCT' has 113 unique values:
[266.9 358.4 354.1 328.1 383.5 251.7 309.7 253.3 212.1 356.6 302.3
469.5
 422.5 374.4 196.6 320.7 264.1 233.2 249.2 350.1 302.4 266.3 203.1
176.5
 295.4 216.7 135.8 321.5 350.4 433.9 149.3 543.2 397.  335.9 403.8
```

```
231.7
 401.9 223.7 339.8 257.7 221.6 427.2 289.  253.8 266.1 183.7 183.9
229.1
 250.4 250.6 339.6 410.5 303.1 378.2 353.3 280.1 191.  200.4 255.9
274.2
 475.6 282.6 325.7 392.3 172.7 178.9 235.6 278.3 220.9 351.5 260.8
142.1
 368.9 221.3 437.  171.  163.8 282.3 265.  164.4 136.2 286.  204.
165.5
 272.1  68.5 308.  323.2 307.8 290.7 431.2 428.4 198.3 294.1 288.9
444.8
 567.9 214.2 319.6 511.7 407.  320.6 240.1 376.4 357.2 343.4 205.2
441.4
 227.2 187.5 259.9 355.5 308.1]
Column 'NOV' has 113 unique values:
[350.8 158.3 157.   33.9  74.4 163.1 219.1  47.9 171.1 280.4 145.7
138.7
 109.9 100.9 302.5 134.3 256.4 295.4 280.1 302.3 136.2 293.7  83.9
162.9
 223.7  88.8 137.6 155.2 158.2 207.  164.3 223.2 126.1  93.4 153.
211.1
 121.   69.5 298.1 287.5 220.5  84.7 223.4 244.1 259.5 273.   32.4
215.6
  71.9 149.2 229.6  49.6  62.2  31.6 178.1 178.2 192.5 206.1 151.9
358.
  85.9  31.5 191.7 131.7 245.4  74.  119.4  80.5  38.3 140.5  84.5
61.
 204.3 286.7 361.7 365.6 261.7 162.3 138.6 127.5 116.5  67.7  74.9
194.7
 216.   67.   92.9 158.8  99.9 287.6 153.8 117.6 182.6  89.9 298.4
135.
  68.1  78.1 181.  137.5  76.4 120.7 184.3 162.8  87.4  55.4 274.4
335.1
 169.7 112.9 154.9  99.5 223.6]
Column 'DEC' has 103 unique values:
[4.840e+01 1.215e+02 5.900e+01 3.300e+00 2.000e-01 8.600e+01 5.280e+01
 1.100e+01 3.230e+01 1.000e-01 8.760e+01 2.200e+01 4.580e+01 1.352e+02
 1.490e+01 8.900e+00 4.160e+01 5.410e+01 5.300e+01 8.200e+00 1.580e+01
 2.510e+01 5.040e+01 9.880e+01 1.620e+01 6.800e+00 5.270e+01 3.940e+01
 8.920e+01 1.065e+02 3.130e+01 4.230e+01 4.900e+00 3.090e+01 1.860e+01
 1.910e+01 2.290e+01 1.020e+01 6.010e+01 8.460e+01 1.179e+02 2.430e+01
 1.800e+01 2.023e+02 4.700e+01 1.920e+01 1.800e+00 8.800e+00 2.320e+01
 6.690e+01 5.100e+00 6.250e+01 1.980e+01 9.100e+00 2.890e+01 7.700e+00
 3.400e+01 2.370e+01 1.750e+01 7.690e+01 1.780e+01 1.555e+02 5.050e+01
 3.110e+01 3.270e+01 6.630e+01 5.700e+00 6.230e+01 1.143e+02 5.380e+01
 3.600e+00 1.990e+01 3.080e+01 6.700e+00 3.900e+01 2.330e+01 3.950e+01
 4.330e+01 1.080e+01 6.910e+01 4.400e+01 9.500e+00 1.311e+02 5.600e+00
 5.200e+00 2.300e+00 3.700e+00 4.620e+01 6.500e+00 8.840e+01 7.940e+01
 1.010e+01 2.100e+00 9.700e+00 2.700e+00 5.640e+01 1.190e+01 1.700e+01
 4.420e+01 4.680e+01 4.950e+01 9.400e+00 4.720e+01]
```

```
Column 'ANNUAL' has 115 unique values:
[3248.6 3326.6 3271.2 3129.7 2741.6 2708.  3671.1 2648.3 3050.2 2848.6
 2726.7 3451.3 2610.8 2899.1 3024.5 2945.3 2704.8 2501.9 3003.3 3303.1
 2719.9 3267.6 3484.7 4226.4 3062.1 2965.4 2994.7 2502.8 3361.6 3018.
 3259.6 3403.  4072.9 2410.7 2498.2 3043.3 2818.2 2634.1 2937.5 3117.8
 3111.1 3050.9 3464.2 2490.  2432.4 3565.5 2998.1 3039.2 2942.6 3146.6
 2705.5 2334.8 2544.9 2938.  3134.7 2798.4 3103.3 2923.1 3746.  3385.5
 4257.8 3375.8 2651.1 2869.1 2342.4 2621.7 2569.1 3392.7 2665.  2703.5
 3076.8 2739.4 2412.5 2767.4 3498.4 2068.8 3047.6 3176.7 2503.  2803.4
 3005.9 2223.3 2320.3 2762.1 2390.5 2093.2 2137.6 2403.5 2422.7 2693.1
 3184.5 3239.5 2717.7 3410.8 2858.8 2610.  3252.4 3151.5 2914.6 2412.6
 2931.1 2507.4 2394.9 2886.1 3031.1 3420.6 3489.6 2524.5 2810.6 3131.8
 3035.1 2151.1 3255.4 3046.4 2600.6]
Column 'JAN-FEB' has 103 unique values:
[73.4  9.3 21.7 26.7 23.4 34.1 23.7 28.8 65.9 28.4  7.3 16.9  8.3  7.6
 40.4  7.8 50.5 47.9 49.2 40.6 47.8 51.9 25.3 22.2 20.5 34.4 54.1 78.6
 42.6 21.6  3.6 19.4 10.3 76.2 32.3 17.7 27.7 79.4 17.2  2.1  7.  98.1
 33.  14.3 49.5 51.3  2.4 53.8 13.1 53.4 35.7 26.3 10.5 19.6 24.3 17.8
 45.  83.1 55.   8.8 10.   9.9 14.4 37.8  9.1 30.5 50.1 10.2  0.3 26.6
  1.6 16.8 18.  32.4  0.9 13.8  0.8  1.7 96.8 67.3  1.4 18.3 19.7 15.3
  3.4 51.4 11.8  3.7  8.  25.5 69.5 44.7 13.3 51.6 26.8  8.6  6.1 31.1
  4.8 66.2 43.9 14.9  8.9]
Column 'MARCH-MAY' has 111 unique values:
[386.2 275.7 336.3 339.4 378.5 230.  328.  283.7 628.3 296.7 249.7
351.1
 295.2 215.  303.1 303.4 240.4 767.  346.8 290.3 399.4 202.3 363.
428.5
 301.5 401.4 254.3 417.6 546.5 277.8 788.1 915.2 246.8 195.4 616.6
371.3
 397.3 302.8 347.6 532.  395.4 624.3 336.  170.2 331.2 325.9 385.5
542.6
 341.5 366.  206.3 407.  698.3 518.  477.2 545.9 504.2 791.  606.
607.1
 323.2 245.2 352.6 330.3 339.6 312.5 364.  447.7 450.6 526.  263.8
365.6
 349.3 231.3 436.9 502.1 181.3 240.9 270.8 230.6  89.9 342.  350.
200.9
 169.7 372.9 341.1 548.3 243.6 261.6 245.6 313.9 527.8 213.  230.2
220.8
 586.2 242.3 483.7 307.5 762.  677.2 338.4 406.7 323.1 360.9 313.5
287.4
 218.5 364.5 465.9]
Column 'JUNE_SEPT' has 113 unique values:
[2122.8 2403.4 2343.  2398.2 1881.5 1943.1 2737.8 2023.6 1940.4 1886.5
 1934.  2453.1 1729.  2066.1 2167.  2170.2 1851.7 1104.3 2025.  2318.2
 1927.5 2231.2 2928.4 3451.3 1995.2 2307.8 2258.9 1640.4 2353.5 1719.7
 2558.  1797.8 2581.9 1653.5 1682.9 1947.5 1877.1 1841.3 1969.4 2162.7
 2031.8 2071.4 2067.  1498.7 1716.5 2485.7 2359.5 2183.7 2094.7 2342.9
 1823.  1477.7 1825.1 2107.4 1849.7 1720.1 1948.7 2831.2 1939.2 3229.3
 2101.6 1848.5 2079.8 1508.9 1593.3 1937.3 2711.4 1870.6 1860.7 2254.6
```

```
 1596.8 1749.2 2188.2 2529.3 1297.1 1788.5 2081.1 1840.5 2077.5 2274.4
 1689.2 1906.9 1951.6 1650.4 1498.4 1347.2 1845.7 1664.7 1638.  2515.6
 2392.5 1823.1 2493.  1933.2 1938.5 2263.4 1661.9 1739.4 1892.  1359.
 1542.6 1669.5 2157.6 2193.8 2688.5 1670.9 1958.9 1928.  2209.1 1535.6
 2561.2 2164.8 1514.7]
Column 'OCT_DEC' has 115 unique values:
[666.1 638.2 570.1 365.3 458.1 500.8 581.7 312.2 415.5 637.  535.7
630.2
 578.3 610.5 514.  463.9 562.1 582.6 582.3 660.6 454.3 585.1 328.6
389.9
 617.9 321.7 280.2 529.4 548.  730.2 420.1 797.7 565.5 434.2 587.6
461.5
 542.  316.1 648.1 605.3 526.7 577.  674.9 622.3 531.4 741.3 263.2
418.7
 302.8 408.4 503.5 456.1 477.8 397.2 576.1 540.6 501.5 404.8 386.3
637.5
 377.6 584.  424.4 535.2 470.9 688.2 277.8 331.  421.3 364.6 321.5
606.4
 399.2 206.6 593.1 538.8 805.4 575.6 448.8 484.1 446.9 302.7 321.8
371.7
 322.8 369.6 619.2 166.6 406.5 487.2 410.  582.1 631.3 552.5 380.9
446.6
 675.7 659.3 640.9 361.5 510.7 651.3 493.1 444.  480.7 541.  456.5
415.7
 523.8 823.3 446.3 309.8 431.8 502.1 611.1]
Column 'FLOOD' has 2 unique values:
[0 1]
Column 'AVGJUNE' has 113 unique values:
[274.8666667  130.3        186.2        366.0666667  283.4
 138.3        256.9666667  197.5333333  234.9        226.6666667
 330.         316.0666667  180.5666667  188.4333333  232.0333333
 306.7333333  234.5666667  154.7666667  212.2666667  321.4333333
 163.0333333  221.0333333  240.8333333  337.2333333  229.6
 187.9666667  240.0666667  196.9        315.5333333  211.0333333
 113.6666667  286.4333333  284.3        143.7666667  206.9333333
 161.8666667  227.2        208.6        202.1333333  265.8666667
 271.2        264.8333333  166.3        183.2666667  306.3333333
 185.3666667  303.4        178.7666667  212.7666667  258.0333333
 192.2333333  113.5        266.1        260.8        251.8
 290.6666667  237.7666667  290.9333333  160.1        335.0666667
  81.63333333 131.1        126.4666667  199.2333333  165.4
 232.1333333  183.5        178.4333333  296.5333333  133.9333333
 205.6666667   88.96666667 288.1333333   65.6        199.8666667
 252.7        194.3        248.6333333  304.1333333  204.0666667
 107.6        280.8666667  276.2333333  199.3        190.8666667
 170.4333333  219.1666667  176.2        365.3666667  273.1
 219.0333333  281.6666667  164.4666667  190.8        181.4
 244.1666667  202.4333333  211.2666667  238.4333333  167.7
 188.9        224.4666667  206.4        160.8        235.3
 156.6333333  146.0666667  222.5        262.8333333  143.4333333
```

```
 347.5666667  151.4666667  187.8666667 ]
Column 'SUB' has 114 unique values:
[649.9 256.4 308.9 862.5 586.9 254.1 669.5 450.  231.5 531.2 809.4
730.9
 342.9 401.1 541.6 721.2 580.8 218.7 389.8 876.6 385.  369.5 642.5
826.3
 430.6 341.3 454.8 508.8 798.6 228.2 410.  305.5 120.5 746.2 374.7
154.3
 348.5 502.  520.7 389.  380.1 621.7 316.1 286.2 496.4 836.  470.4
697.9
  96.3 396.3 625.6 362.1 285.1 618.8 238.2 404.1 490.8 359.8 525.6
59.7
 504.7 227.5 236.2 284.6 383.2 401.  296.8 606.4 323.1 246.2 572.1
34.2
 497.1  45.4 702.2 121.  293.2 361.3 455.2 640.6 746.1 464.  246.8
758.
 574.5 471.2 464.3 354.1 488.1  40.1 982.7 600.9 498.1 703.7 137.8
410.6
 580.9 154.1 509.3 476.7 172.3 475.7  62.5 484.4  38.8 513.2 388.7
246.6
 476.9 664.3 335.  923.4 203.4 361.8]
```

```python
#Create function to check statistical summary of the dataset
def stat_summary (df):
    statistics_sum = df.describe()
    return statistics_sum
#print statistical summary of data
summary = stat_summary (df)
summary
```

```
              YEAR         JAN         FEB       MARCH       APRIL  \
count   115.000000  115.000000  115.000000  115.000000  115.000000
mean   1958.000000   12.246957   15.496522   36.814783  110.573913
std      33.341666   15.538923   16.206572   30.324601   44.673971
min    1901.000000    0.000000    0.000000    0.100000   13.100000
25%    1929.500000    2.250000    4.700000   18.100000   74.800000
50%    1958.000000    6.000000    8.400000   28.300000  109.800000
75%    1986.500000   17.750000   21.400000   50.000000  136.000000
max    2015.000000   83.500000   79.000000  217.200000  238.000000

              MAY         JUNE         JULY          AUG
SEPT  ...  \
count  115.000000   115.000000   115.000000   115.000000
115.000000  ...
mean   229.881739   654.302609   700.953043   421.977391
245.619130  ...
std    149.271697   187.642791   225.294102   159.693779
122.130976  ...
min     53.400000   196.800000   167.500000   178.600000
41.300000  ...
```

```
25%    124.350000   539.000000   540.700000   315.550000
152.550000  ...
50%    185.400000   633.100000   696.000000   385.200000
223.900000  ...
75%    277.250000   791.500000   832.150000   495.300000
333.400000  ...
max    738.800000  1098.200000  1526.500000  1199.200000
526.700000  ...

              NOV          DEC        ANNUAL     JAN-FEB   MARCH-MAY  \
count  115.000000  115.000000   115.000000  115.000000  115.000000
mean   163.560000   39.950435  2925.487826   27.739130  377.253913
std     83.882421   37.049051   422.112193   22.361032  151.091850
min     31.500000    0.100000  2068.800000    0.300000   89.900000
25%     93.150000   10.150000  2627.900000   10.250000  276.750000
50%    153.800000   31.100000  2937.500000   20.500000  342.000000
75%    219.800000   53.950000  3164.100000   41.600000  442.300000
max    365.600000  202.300000  4257.800000   98.100000  915.200000

         JUNE_SEPT     OCT_DEC        FLOOD     AVGJUNE         SUB
count   115.000000  115.000000  115.000000  115.000000  115.000000
mean   2022.840870  497.636522    0.139130  218.100870  439.801739
std     386.254397  129.860643    0.347597   62.547597  210.438813
min    1104.300000  166.600000    0.000000   65.600000   34.200000
25%    1768.850000  407.450000    0.000000  179.666667  295.000000
50%    1948.700000  501.500000    0.000000  211.033333  430.600000
75%    2242.900000  584.550000    0.000000  263.833333  577.650000
max    3451.300000  823.300000    1.000000  366.066667  982.700000

[8 rows x 21 columns]
```

```
#create function to find columns with Categorical Values

def categorcal_columns (df):
    cat_cols=
df.select_dtypes(exclude=['int','float']).columns.tolist()
    return cat_cols
#print categorical columns
categorical = categorcal_columns (df)
categorical
```

```
['SUBDIVISION']
```

```
#create function to find columns with Numerical Values

def numerical_columns (df):
    num_cols =
df.select_dtypes(include=['int','float']).columns.tolist()
    return num_cols
#print numerical columns
```

```
numerical = numerical_columns (df)
numerical
```

```
['YEAR',
 'JAN',
 'FEB',
 'MARCH',
 'APRIL',
 'MAY',
 'JUNE',
 'JULY',
 'AUG',
 'SEPT',
 'OCT',
 'NOV',
 'DEC',
 'ANNUAL',
 'JAN-FEB',
 'MARCH-MAY',
 'JUNE_SEPT',
 'OCT_DEC',
 'FLOOD',
 'AVGJUNE',
 'SUB']
```

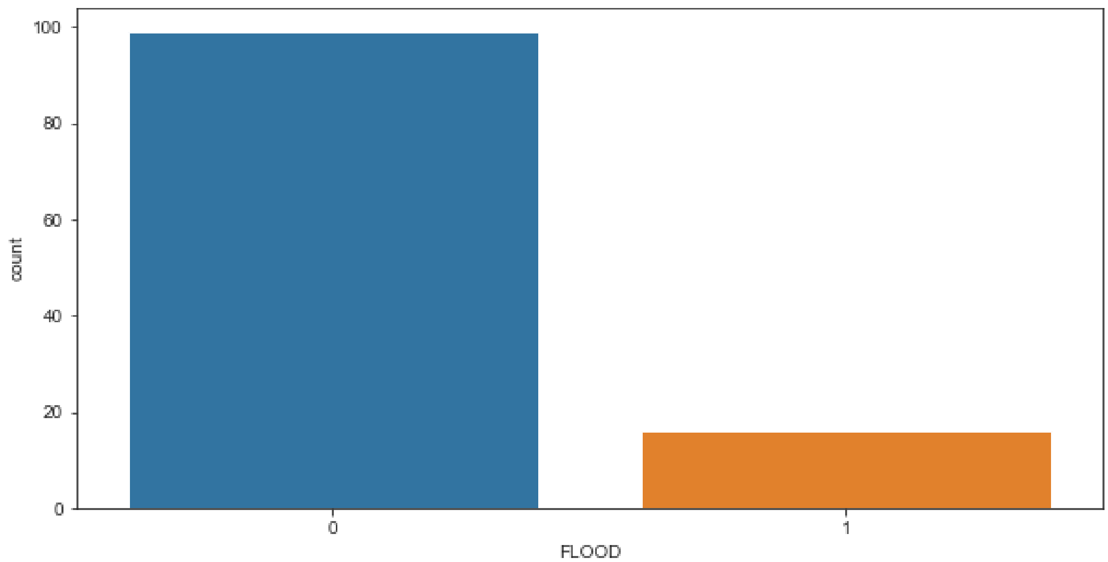
```
#Removing extra spaces from the features Using strip()
df.columns = df.columns.str.strip()
```

Target Class Exploration

```
#Target Class Exploration
#Estimating the data-points for target class
df['FLOOD'].value_counts()
```

```
0    99
1    16
Name: FLOOD, dtype: int64
```

```
#Distribution of Flood in dataset
#countplot showing numbers of 0 and 1

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

sns.set_style('ticks')
plt.figure(figsize = (10, 5))
sns.countplot(df['FLOOD']);
plt.show()
```

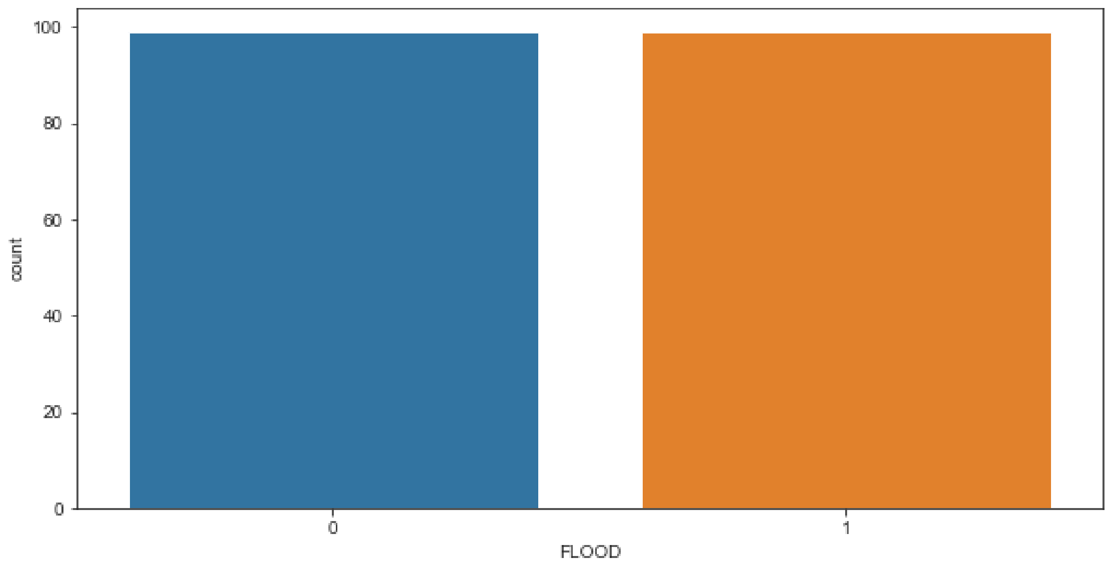



```
#Evaluating if dataset is balanced or not
percentage_1 = (df['FLOOD'].value_counts()[1] /
                  (df['FLOOD'].value_counts()[1] +
df['FLOOD'].value_counts()[0]))*100
percentage_0 = (df['FLOOD'].value_counts()[0] /
                 (df['FLOOD'].value_counts()[1] +
df['FLOOD'].value_counts()[0]))*100

print(f"Percentage of 1 according to dataset is {round(percentage_1)}
%.\nand")
print(f"Percentage of 0 according to dataset is {round(percentage_0)}
%.\nand")

#Threshold value to evaluate the difference between the dataset will
be greator than the Threshold value
Threshold = 5
#zero difference is acceptable

if(abs(percentage_1- percentage_0) > Threshold):
    print("Imbalanced dataset")
else:
    print("Balanced dataset")

Percentage of 1 according to dataset is 14%.
and
Percentage of 0 according to dataset is 86%.
and
Imbalanced dataset

#Minority Oversampling Technique is used to oversample the minority
class i.e 1
```

```python
from sklearn.utils import resample
# Separate majority and minority classes
df_majority = df[df.FLOOD==0]
df_minority = df[df.FLOOD==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with 
replacement
                                 n_samples=99,    # to match majority 
class
                                 random_state=123) # reproducible 
results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled.FLOOD.value_counts()
```

```
0    99
1    99
Name: FLOOD, dtype: int64
```

```python
#Distribution of Flood in balanced dataset
#countplot showing numbers of 0 and 1

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

sns.set_style('ticks')
plt.figure(figsize = (10, 5))
sns.countplot(df_upsampled['FLOOD']);
plt.show()
```



#Outlier detection

```
#Outlier detection

# calculate IQR score
#where Q3 is the 75th percentile of the data and Q1 is the 25th
percentile of the data.

Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

print(IQR)

# IQR of each column

YEAR          57.000000
JAN           15.500000
FEB           16.700000
MARCH         31.900000
APRIL         61.200000
MAY          152.900000
JUNE         252.500000
JULY         291.450000
AUG          179.750000
SEPT         180.850000
OCT          132.150000
NOV          126.650000
DEC           43.800000
ANNUAL       536.200000
JAN-FEB       31.350000
MARCH-MAY    165.550000
```

```
JUNE_SEPT    474.050000
OCT_DEC      177.100000
FLOOD          0.000000
AVGJUNE       84.166667
SUB          282.650000
dtype: float64
```

Feature Scaling

```
# Standardization
#Standardize input variables

from sklearn.preprocessing import StandardScaler

# Create a StandardScaler object
scaler = StandardScaler()
import warnings
from pandas.core.common import SettingWithCopyWarning
warnings.simplefilter(action="ignore",
category=SettingWithCopyWarning)

# Standardize variables #MARCH-MAY #Average of 10days in June i.e
AVGJUNE
#Difference of Rainfall from May to June i.e SUB in the dataframe

df_upsampled[['MARCH-MAY', 'AVGJUNE', 'SUB']] =
scaler.fit_transform(df_upsampled[['MARCH-MAY', 'AVGJUNE', 'SUB']])

#check standardized columns
df_upsampled
```

```
   SUBDIVISION  YEAR   JAN   FEB  MARCH  APRIL    MAY    JUNE    JULY
AUG  \
0       KERALA  1901  28.7  44.7   51.6  160.0  174.7   824.6   743.0
357.5
2       KERALA  1903   3.2  18.6    3.1   83.6  249.7   558.6  1022.5
420.2
3       KERALA  1904  23.7   3.0   32.2   71.5  235.7  1098.2   725.5
351.8
4       KERALA  1905   1.2  22.3    9.4  105.9  263.3   850.2   520.5
293.6
5       KERALA  1906  26.7   7.4    9.9   59.4  160.8   414.9   954.2
442.8
..         ...   ...   ...   ...    ...    ...    ...     ...     ...
...
30      KERALA  1931   3.3   0.3   19.2  126.9  131.7   541.7   653.9
1199.2
45      KERALA  1946   1.8   5.4  108.4  139.8   83.0   919.0   671.7
695.6
60      KERALA  1961  13.7  31.3   11.4   94.1  500.5  1005.2  1146.5
```

```
678.3
11      KERALA  1912   1.9  15.0   11.2  122.7  217.3   948.2   833.6
534.4
23      KERALA  1924  19.3   2.9   66.6  111.0  185.4  1011.7  1526.5
624.0

    ...    NOV    DEC  ANNUAL  JAN-FEB  MARCH-MAY  JUNE_SEPT  OCT_DEC
FLOOD  \
0   ...  350.8   48.4  3248.6     73.4   0.055908     2122.8    666.1
0
2   ...  157.0   59.0  3271.2     21.7  -0.247413     2343.0    570.1
0
3   ...   33.9    3.3  3129.7     26.7  -0.228570     2398.2    365.3
0
4   ...   74.4    0.2  2741.6     23.4   0.009103     1881.5    458.1
0
5   ...  163.1   86.0  2708.0     34.1  -0.893566     1943.1    500.8
0
..  ...    ...    ...     ...      ...        ...        ...      ...
...
30  ...  164.3  106.5  3259.6      3.6  -0.603010     2558.0    420.1
1
45  ...  273.0  202.3  3565.5      7.3  -0.278414     2485.7    741.3
1
60  ...   85.9   17.5  4257.8     45.0   1.391978     3229.3    377.6
1
11  ...  138.7   22.0  3451.3     16.9  -0.157450     2453.1    630.2
1
23  ...  162.9   50.4  4226.4     22.2  -0.085115     3451.3    389.9
1

     AVGJUNE       SUB
0   0.478446  0.575789
2  -0.824234 -0.864942
3   1.818346  1.474027
4   0.603817  0.309613
5  -1.527976 -1.096473
..       ...       ...
30 -0.906999 -0.437793
45  0.940751  1.362064
60  1.362897 -0.037684
11  1.083752  0.918015
23  1.394730  1.321082

[198 rows x 22 columns]
```

Train_Test_Split on balanced Data

```
from sklearn.model_selection import train_test_split

# Split data into input (X) and output (y)

# Select the input and output features

X = df_upsampled[['MARCH-MAY', 'AVGJUNE', 'SUB']] # select input
features
y = df_upsampled['FLOOD'] # select the output feature

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.34, random_state=42)

#test_size set split as 66:34 with training set as 66 and test set as
34.
#set random_state to 42 to ensure that the results are reproducible.

#training set

X_train.shape

(130, 3)

#testng set

X_test.shape

(68, 3)
```

Logistic Regression

```
#Importing libraries

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

# Create the logistic Regression model using dafault hyperparameters

lr = LogisticRegression()

#fitting Logistic Regression to the training set

lr=lr.fit (X_train, y_train)
```

```python
#Predicting output with the test data
y_pred = lr.predict (X_test)
y_pred
```

```
array([0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
0,
       0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
1,
       0, 1], dtype=int64)
```

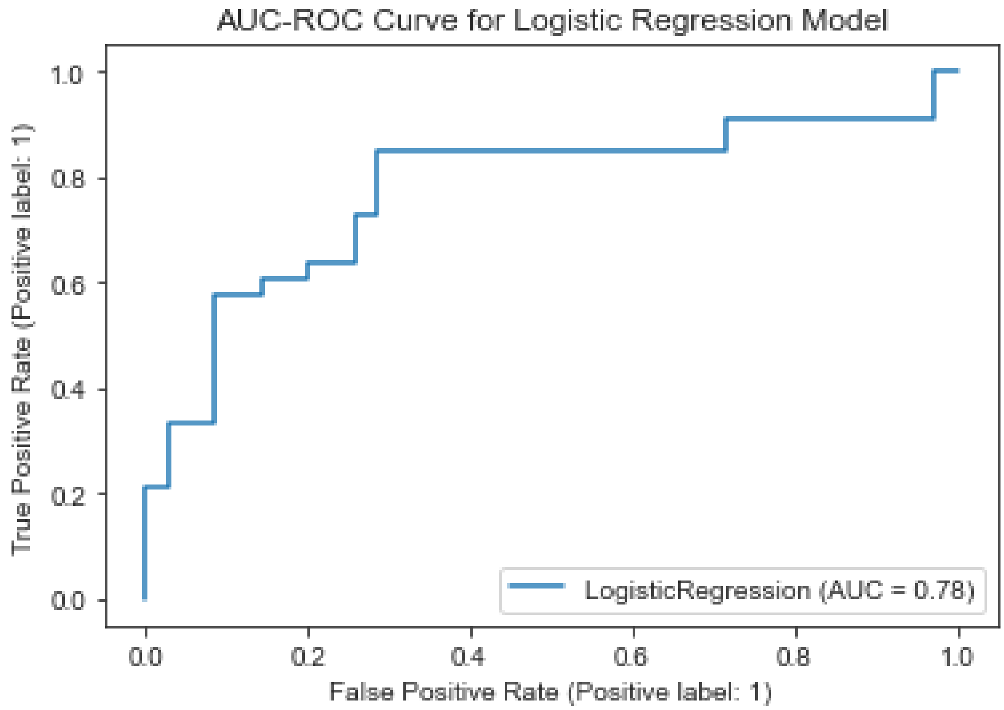
```python
#Calculating test accuracy
#testing the model
y_pred

#importing accuracy score
from sklearn.metrics import accuracy_score

#printing the accuracy of the model
accuracy = (round(accuracy_score(y_test, y_pred) * 100, 0))
print ("Accuracy:", accuracy)
```

```
Accuracy: 74.0
```

```python
# Predict probabilities of the positive class for the test set
y_prob = lr.predict_proba(X_test)[:, 1]

# Calculate AUC-ROC score
auc_roc = roc_auc_score(y_test, y_prob)

print('AUC-ROC score:', auc_roc)
```

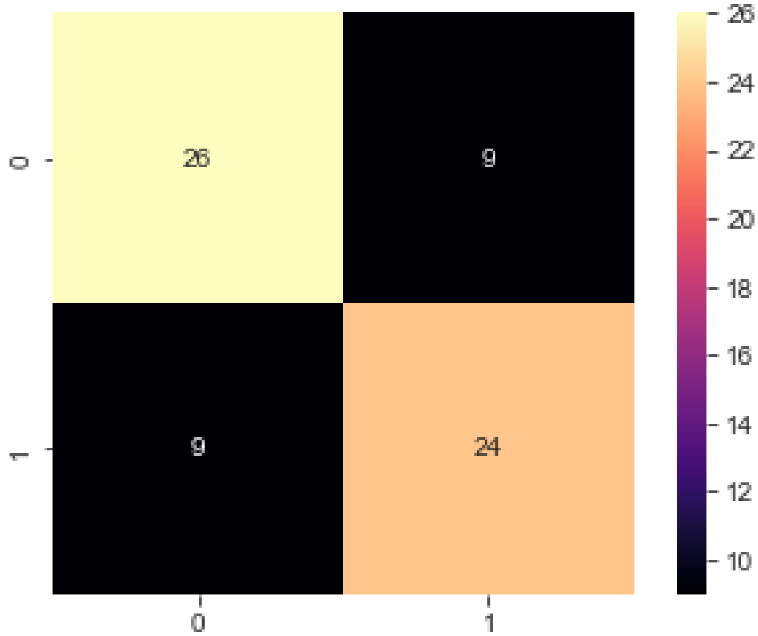
```
AUC-ROC score: 0.7757575757575758
```

```python
#plot AUC-ROC
from sklearn.metrics import plot_roc_curve

# Plot the AUC-ROC curve for the logistic regression model
plot_roc_curve(lr, X_test, y_test)
plt.title('AUC-ROC Curve for Logistic Regression Model')
plt.show()
```




```
#Confusion matrics
#A confusion matrix is a technique for summarizing the performance of 
a classification algorithm.
#Importing confusion matrics
from sklearn.metrics import confusion_matrix
classifier=lr
conf_mat = confusion_matrix(y_test, classifier.predict(X_test))
sns.heatmap(conf_mat, square=True, annot=True, cmap='magma', fmt='d', 
cbar=True)
```

```
<AxesSubplot:>
```




```
predictions = lr.predict(X_test)
#Importing classifiation_report, confusion_matrix
from sklearn.metrics import classification_report,confusion_matrix
#printing confusion matrix from the y_test values and predictions
print(confusion_matrix(y_test,predictions))
```

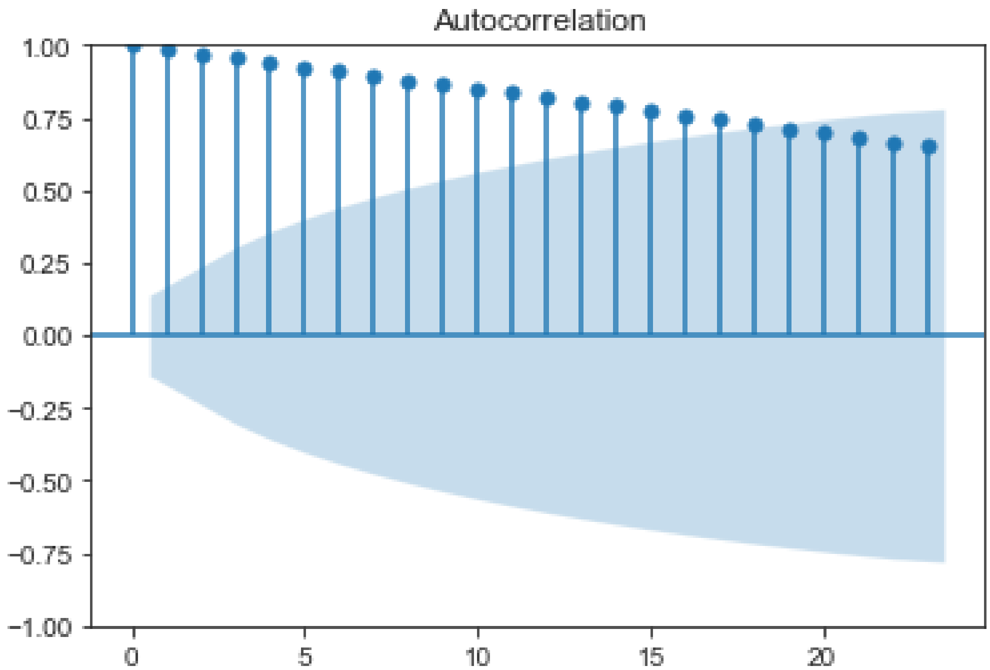
```
[[26  9]
 [ 9 24]]
```

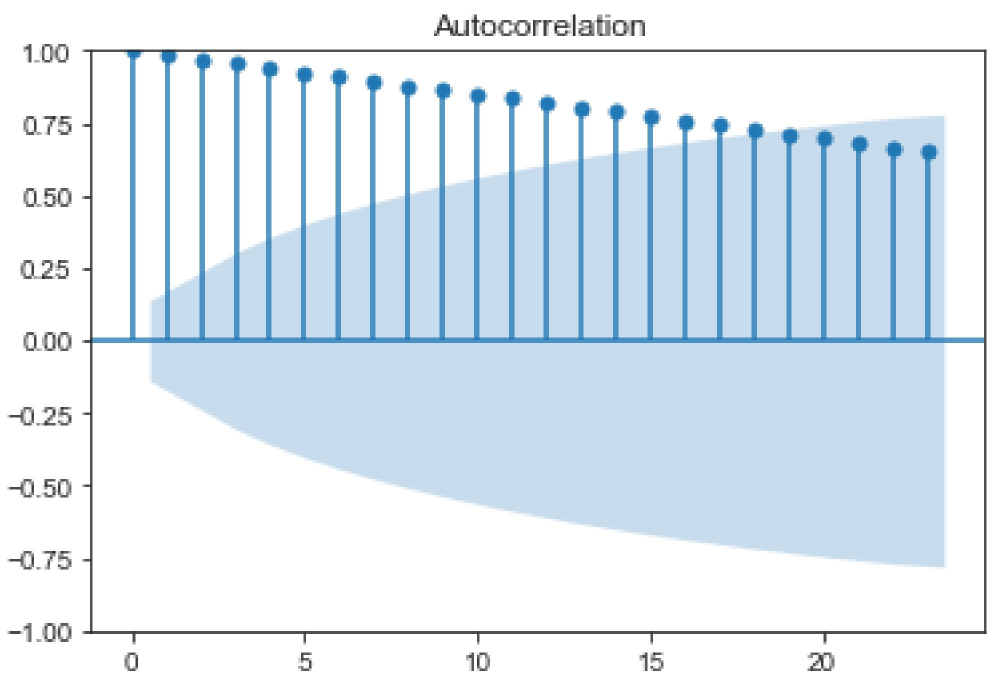
```
#printing classification report from the y_test values and predictions
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.74      0.74      0.74        35
           1       0.73      0.73      0.73        33

    accuracy                           0.74        68
   macro avg       0.74      0.74      0.74        68
weighted avg       0.74      0.74      0.74        68
```

LTSM (Long short-term memory networks) Modelling

```
#determine the lag value at which the autocorrelation drops off
from statsmodels.graphics.tsaplots import plot_acf
# Plot ACF for the 'FLOOD' column in the balanced DataFrame
plot_acf(df_upsampled['FLOOD'], fft=False)
```





Train_Test_split on balanced data

```
# Split data into input (X) and output (y)

# Select the input and output features

X = df_upsampled[['MARCH-MAY', 'AVGJUNE', 'SUB']] # select input
```

```
features
y = df_upsampled['FLOOD'] # select the output feature

# Split data into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.34, random_state=42)

#test_size set split as 66:34 with training set as 66 and test set as
34.
#set random_state to 42 to ensure that the results are reproducible.

# Reshape the input data for the LSTM layer
X_train = X_train.values.reshape((X_train.shape[0], 1,
X_train.shape[1]))
X_test = X_test.values.reshape((X_test.shape[0], 1, X_test.shape[1]))

# Define the LSTM model architecture

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score, roc_auc_score
from tensorflow.keras.optimizers import Adam


model = Sequential()
model.add(LSTM(60, activation='relu', input_shape=(1,
X_train.shape[2])))
model.add(Dense(1))

learning_rate = 0.001
optimizer = Adam(learning_rate=learning_rate)
model.compile(loss='mse', optimizer=optimizer,metrics=['accuracy'])

# print the model summary
print(model.summary())

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 60)                15360

 dense (Dense)               (None, 1)                 61

=================================================================
Total params: 15,421
Trainable params: 15,421
Non-trainable params: 0
```

None

```python
# Train the model using the training set
history = model.fit(X_train, y_train, epochs=100, batch_size=16,
validation_data=(X_test, y_test), verbose=2, shuffle=False)
```

```
Epoch 1/100
9/9 - 2s - loss: 0.4624 - accuracy: 0.4923 - val_loss: 0.4186 -
val_accuracy: 0.5147 - 2s/epoch - 189ms/step
Epoch 2/100
9/9 - 0s - loss: 0.4172 - accuracy: 0.4923 - val_loss: 0.3821 -
val_accuracy: 0.5147 - 55ms/epoch - 6ms/step
Epoch 3/100
9/9 - 0s - loss: 0.3764 - accuracy: 0.4923 - val_loss: 0.3485 -
val_accuracy: 0.5147 - 56ms/epoch - 6ms/step
Epoch 4/100
9/9 - 0s - loss: 0.3388 - accuracy: 0.4923 - val_loss: 0.3173 -
val_accuracy: 0.5147 - 55ms/epoch - 6ms/step
Epoch 5/100
9/9 - 0s - loss: 0.3042 - accuracy: 0.5000 - val_loss: 0.2885 -
val_accuracy: 0.5294 - 60ms/epoch - 7ms/step
Epoch 6/100
9/9 - 0s - loss: 0.2724 - accuracy: 0.5462 - val_loss: 0.2624 -
val_accuracy: 0.5588 - 69ms/epoch - 8ms/step
Epoch 7/100
9/9 - 0s - loss: 0.2441 - accuracy: 0.5385 - val_loss: 0.2395 -
val_accuracy: 0.6029 - 57ms/epoch - 6ms/step
Epoch 8/100
9/9 - 0s - loss: 0.2201 - accuracy: 0.6385 - val_loss: 0.2209 -
val_accuracy: 0.6029 - 54ms/epoch - 6ms/step
Epoch 9/100
9/9 - 0s - loss: 0.2014 - accuracy: 0.6923 - val_loss: 0.2073 -
val_accuracy: 0.6324 - 58ms/epoch - 6ms/step
Epoch 10/100
9/9 - 0s - loss: 0.1883 - accuracy: 0.6846 - val_loss: 0.1987 -
val_accuracy: 0.7647 - 56ms/epoch - 6ms/step
Epoch 11/100
9/9 - 0s - loss: 0.1804 - accuracy: 0.7692 - val_loss: 0.1943 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 12/100
9/9 - 0s - loss: 0.1764 - accuracy: 0.7846 - val_loss: 0.1925 -
val_accuracy: 0.7353 - 75ms/epoch - 8ms/step
Epoch 13/100
9/9 - 0s - loss: 0.1744 - accuracy: 0.7769 - val_loss: 0.1917 -
val_accuracy: 0.7353 - 105ms/epoch - 12ms/step
Epoch 14/100
9/9 - 0s - loss: 0.1730 - accuracy: 0.8154 - val_loss: 0.1911 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 15/100
9/9 - 0s - loss: 0.1716 - accuracy: 0.8154 - val_loss: 0.1903 -
```

```
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 16/100
9/9 - 0s - loss: 0.1700 - accuracy: 0.8077 - val_loss: 0.1895 -
val_accuracy: 0.7500 - 64ms/epoch - 7ms/step
Epoch 17/100
9/9 - 0s - loss: 0.1684 - accuracy: 0.8000 - val_loss: 0.1887 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 18/100
9/9 - 0s - loss: 0.1669 - accuracy: 0.7923 - val_loss: 0.1879 -
val_accuracy: 0.7500 - 66ms/epoch - 7ms/step
Epoch 19/100
9/9 - 0s - loss: 0.1655 - accuracy: 0.7923 - val_loss: 0.1872 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 20/100
9/9 - 0s - loss: 0.1643 - accuracy: 0.7846 - val_loss: 0.1867 -
val_accuracy: 0.7500 - 57ms/epoch - 6ms/step
Epoch 21/100
9/9 - 0s - loss: 0.1631 - accuracy: 0.7846 - val_loss: 0.1862 -
val_accuracy: 0.7500 - 69ms/epoch - 8ms/step
Epoch 22/100
9/9 - 0s - loss: 0.1621 - accuracy: 0.7846 - val_loss: 0.1858 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 23/100
9/9 - 0s - loss: 0.1611 - accuracy: 0.7846 - val_loss: 0.1854 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 24/100
9/9 - 0s - loss: 0.1602 - accuracy: 0.8000 - val_loss: 0.1851 -
val_accuracy: 0.7500 - 68ms/epoch - 8ms/step
Epoch 25/100
9/9 - 0s - loss: 0.1594 - accuracy: 0.8000 - val_loss: 0.1848 -
val_accuracy: 0.7500 - 54ms/epoch - 6ms/step
Epoch 26/100
9/9 - 0s - loss: 0.1586 - accuracy: 0.7923 - val_loss: 0.1846 -
val_accuracy: 0.7500 - 54ms/epoch - 6ms/step
Epoch 27/100
9/9 - 0s - loss: 0.1579 - accuracy: 0.7846 - val_loss: 0.1843 -
val_accuracy: 0.7500 - 66ms/epoch - 7ms/step
Epoch 28/100
9/9 - 0s - loss: 0.1572 - accuracy: 0.7846 - val_loss: 0.1840 -
val_accuracy: 0.7500 - 68ms/epoch - 8ms/step
Epoch 29/100
9/9 - 0s - loss: 0.1566 - accuracy: 0.7846 - val_loss: 0.1837 -
val_accuracy: 0.7353 - 55ms/epoch - 6ms/step
Epoch 30/100
9/9 - 0s - loss: 0.1560 - accuracy: 0.7846 - val_loss: 0.1835 -
val_accuracy: 0.7353 - 55ms/epoch - 6ms/step
Epoch 31/100
9/9 - 0s - loss: 0.1554 - accuracy: 0.7846 - val_loss: 0.1832 -
val_accuracy: 0.7353 - 57ms/epoch - 6ms/step
Epoch 32/100
```

```
9/9 - 0s - loss: 0.1549 - accuracy: 0.7846 - val_loss: 0.1829 -
val_accuracy: 0.7647 - 64ms/epoch - 7ms/step
Epoch 33/100
9/9 - 0s - loss: 0.1544 - accuracy: 0.8077 - val_loss: 0.1827 -
val_accuracy: 0.7647 - 57ms/epoch - 6ms/step
Epoch 34/100
9/9 - 0s - loss: 0.1538 - accuracy: 0.8077 - val_loss: 0.1824 -
val_accuracy: 0.7647 - 58ms/epoch - 6ms/step
Epoch 35/100
9/9 - 0s - loss: 0.1533 - accuracy: 0.8000 - val_loss: 0.1821 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 36/100
9/9 - 0s - loss: 0.1528 - accuracy: 0.8000 - val_loss: 0.1818 -
val_accuracy: 0.7500 - 62ms/epoch - 7ms/step
Epoch 37/100
9/9 - 0s - loss: 0.1523 - accuracy: 0.8000 - val_loss: 0.1816 -
val_accuracy: 0.7500 - 70ms/epoch - 8ms/step
Epoch 38/100
9/9 - 0s - loss: 0.1519 - accuracy: 0.8000 - val_loss: 0.1813 -
val_accuracy: 0.7500 - 58ms/epoch - 6ms/step
Epoch 39/100
9/9 - 0s - loss: 0.1514 - accuracy: 0.8000 - val_loss: 0.1810 -
val_accuracy: 0.7500 - 57ms/epoch - 6ms/step
Epoch 40/100
9/9 - 0s - loss: 0.1509 - accuracy: 0.8000 - val_loss: 0.1808 -
val_accuracy: 0.7500 - 57ms/epoch - 6ms/step
Epoch 41/100
9/9 - 0s - loss: 0.1505 - accuracy: 0.8000 - val_loss: 0.1806 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 42/100
9/9 - 0s - loss: 0.1501 - accuracy: 0.8077 - val_loss: 0.1804 -
val_accuracy: 0.7500 - 62ms/epoch - 7ms/step
Epoch 43/100
9/9 - 0s - loss: 0.1497 - accuracy: 0.8077 - val_loss: 0.1801 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 44/100
9/9 - 0s - loss: 0.1493 - accuracy: 0.8077 - val_loss: 0.1799 -
val_accuracy: 0.7500 - 63ms/epoch - 7ms/step
Epoch 45/100
9/9 - 0s - loss: 0.1489 - accuracy: 0.8077 - val_loss: 0.1796 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 46/100
9/9 - 0s - loss: 0.1485 - accuracy: 0.8077 - val_loss: 0.1793 -
val_accuracy: 0.7500 - 61ms/epoch - 7ms/step
Epoch 47/100
9/9 - 0s - loss: 0.1481 - accuracy: 0.8077 - val_loss: 0.1791 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 48/100
9/9 - 0s - loss: 0.1477 - accuracy: 0.8077 - val_loss: 0.1788 -
val_accuracy: 0.7500 - 64ms/epoch - 7ms/step
```

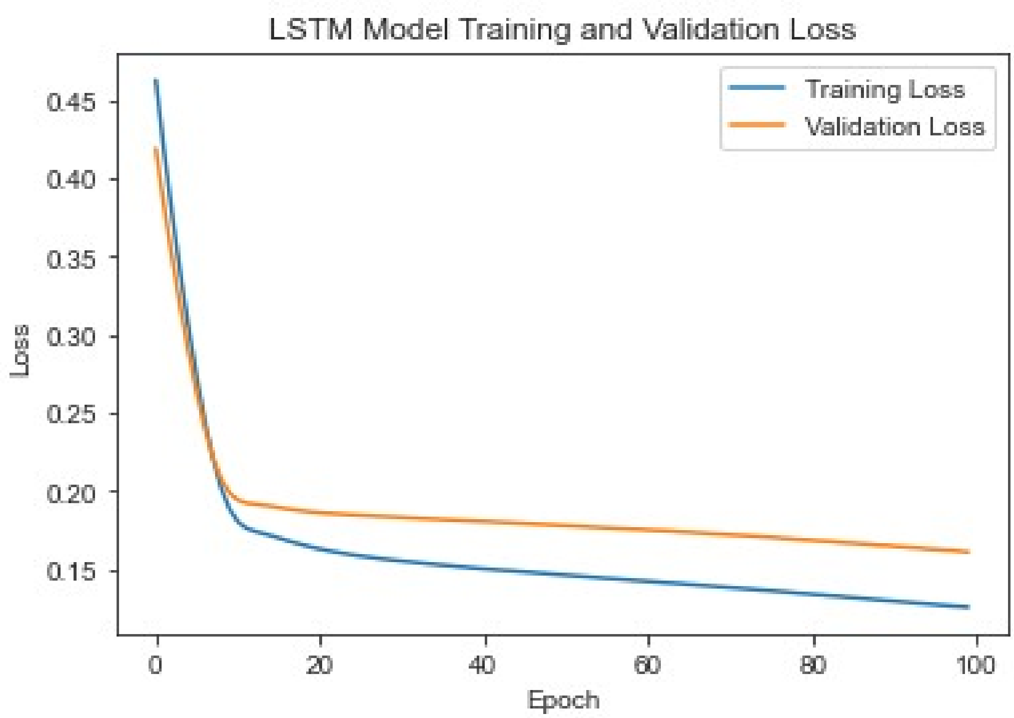
```
Epoch 49/100
9/9 - 0s - loss: 0.1473 - accuracy: 0.8077 - val_loss: 0.1786 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 50/100
9/9 - 0s - loss: 0.1469 - accuracy: 0.8077 - val_loss: 0.1782 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 51/100
9/9 - 0s - loss: 0.1464 - accuracy: 0.8077 - val_loss: 0.1780 -
val_accuracy: 0.7500 - 58ms/epoch - 6ms/step
Epoch 52/100
9/9 - 0s - loss: 0.1460 - accuracy: 0.8077 - val_loss: 0.1777 -
val_accuracy: 0.7500 - 76ms/epoch - 8ms/step
Epoch 53/100
9/9 - 0s - loss: 0.1456 - accuracy: 0.8077 - val_loss: 0.1774 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 54/100
9/9 - 0s - loss: 0.1452 - accuracy: 0.8077 - val_loss: 0.1771 -
val_accuracy: 0.7500 - 60ms/epoch - 7ms/step
Epoch 55/100
9/9 - 0s - loss: 0.1448 - accuracy: 0.8077 - val_loss: 0.1768 -
val_accuracy: 0.7500 - 54ms/epoch - 6ms/step
Epoch 56/100
9/9 - 0s - loss: 0.1444 - accuracy: 0.8077 - val_loss: 0.1765 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 57/100
9/9 - 0s - loss: 0.1440 - accuracy: 0.8077 - val_loss: 0.1762 -
val_accuracy: 0.7500 - 61ms/epoch - 7ms/step
Epoch 58/100
9/9 - 0s - loss: 0.1436 - accuracy: 0.8077 - val_loss: 0.1760 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 59/100
9/9 - 0s - loss: 0.1432 - accuracy: 0.8077 - val_loss: 0.1758 -
val_accuracy: 0.7500 - 54ms/epoch - 6ms/step
Epoch 60/100
9/9 - 0s - loss: 0.1429 - accuracy: 0.8077 - val_loss: 0.1756 -
val_accuracy: 0.7500 - 58ms/epoch - 6ms/step
Epoch 61/100
9/9 - 0s - loss: 0.1425 - accuracy: 0.8077 - val_loss: 0.1752 -
val_accuracy: 0.7500 - 57ms/epoch - 6ms/step
Epoch 62/100
9/9 - 0s - loss: 0.1421 - accuracy: 0.8077 - val_loss: 0.1749 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 63/100
9/9 - 0s - loss: 0.1417 - accuracy: 0.8077 - val_loss: 0.1747 -
val_accuracy: 0.7500 - 54ms/epoch - 6ms/step
Epoch 64/100
9/9 - 0s - loss: 0.1413 - accuracy: 0.8077 - val_loss: 0.1743 -
val_accuracy: 0.7500 - 53ms/epoch - 6ms/step
Epoch 65/100
9/9 - 0s - loss: 0.1409 - accuracy: 0.8077 - val_loss: 0.1740 -
```

```
val_accuracy: 0.7500 - 50ms/epoch - 6ms/step
Epoch 66/100
9/9 - 0s - loss: 0.1405 - accuracy: 0.8154 - val_loss: 0.1738 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 67/100
9/9 - 0s - loss: 0.1401 - accuracy: 0.8154 - val_loss: 0.1734 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 68/100
9/9 - 0s - loss: 0.1397 - accuracy: 0.8154 - val_loss: 0.1731 -
val_accuracy: 0.7500 - 57ms/epoch - 6ms/step
Epoch 69/100
9/9 - 0s - loss: 0.1393 - accuracy: 0.8154 - val_loss: 0.1727 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 70/100
9/9 - 0s - loss: 0.1389 - accuracy: 0.8154 - val_loss: 0.1725 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 71/100
9/9 - 0s - loss: 0.1385 - accuracy: 0.8154 - val_loss: 0.1722 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 72/100
9/9 - 0s - loss: 0.1381 - accuracy: 0.8154 - val_loss: 0.1718 -
val_accuracy: 0.7500 - 51ms/epoch - 6ms/step
Epoch 73/100
9/9 - 0s - loss: 0.1377 - accuracy: 0.8154 - val_loss: 0.1715 -
val_accuracy: 0.7500 - 58ms/epoch - 6ms/step
Epoch 74/100
9/9 - 0s - loss: 0.1373 - accuracy: 0.8154 - val_loss: 0.1712 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 75/100
9/9 - 0s - loss: 0.1369 - accuracy: 0.8154 - val_loss: 0.1708 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 76/100
9/9 - 0s - loss: 0.1365 - accuracy: 0.8231 - val_loss: 0.1705 -
val_accuracy: 0.7500 - 49ms/epoch - 5ms/step
Epoch 77/100
9/9 - 0s - loss: 0.1360 - accuracy: 0.8231 - val_loss: 0.1701 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 78/100
9/9 - 0s - loss: 0.1356 - accuracy: 0.8231 - val_loss: 0.1696 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 79/100
9/9 - 0s - loss: 0.1351 - accuracy: 0.8231 - val_loss: 0.1693 -
val_accuracy: 0.7500 - 52ms/epoch - 6ms/step
Epoch 80/100
9/9 - 0s - loss: 0.1347 - accuracy: 0.8231 - val_loss: 0.1689 -
val_accuracy: 0.7500 - 60ms/epoch - 7ms/step
Epoch 81/100
9/9 - 0s - loss: 0.1342 - accuracy: 0.8231 - val_loss: 0.1686 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 82/100
```

```
9/9 - 0s - loss: 0.1338 - accuracy: 0.8231 - val_loss: 0.1682 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 83/100
9/9 - 0s - loss: 0.1334 - accuracy: 0.8231 - val_loss: 0.1680 -
val_accuracy: 0.7500 - 50ms/epoch - 6ms/step
Epoch 84/100
9/9 - 0s - loss: 0.1330 - accuracy: 0.8231 - val_loss: 0.1675 -
val_accuracy: 0.7500 - 52ms/epoch - 6ms/step
Epoch 85/100
9/9 - 0s - loss: 0.1325 - accuracy: 0.8231 - val_loss: 0.1670 -
val_accuracy: 0.7500 - 58ms/epoch - 6ms/step
Epoch 86/100
9/9 - 0s - loss: 0.1320 - accuracy: 0.8231 - val_loss: 0.1668 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
Epoch 87/100
9/9 - 0s - loss: 0.1317 - accuracy: 0.8231 - val_loss: 0.1663 -
val_accuracy: 0.7500 - 57ms/epoch - 6ms/step
Epoch 88/100
9/9 - 0s - loss: 0.1312 - accuracy: 0.8231 - val_loss: 0.1660 -
val_accuracy: 0.7500 - 51ms/epoch - 6ms/step
Epoch 89/100
9/9 - 0s - loss: 0.1308 - accuracy: 0.8231 - val_loss: 0.1655 -
val_accuracy: 0.7500 - 52ms/epoch - 6ms/step
Epoch 90/100
9/9 - 0s - loss: 0.1302 - accuracy: 0.8231 - val_loss: 0.1652 -
val_accuracy: 0.7500 - 52ms/epoch - 6ms/step
Epoch 91/100
9/9 - 0s - loss: 0.1299 - accuracy: 0.8231 - val_loss: 0.1647 -
val_accuracy: 0.7500 - 60ms/epoch - 7ms/step
Epoch 92/100
9/9 - 0s - loss: 0.1294 - accuracy: 0.8231 - val_loss: 0.1645 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 93/100
9/9 - 0s - loss: 0.1290 - accuracy: 0.8231 - val_loss: 0.1640 -
val_accuracy: 0.7500 - 54ms/epoch - 6ms/step
Epoch 94/100
9/9 - 0s - loss: 0.1285 - accuracy: 0.8231 - val_loss: 0.1637 -
val_accuracy: 0.7500 - 53ms/epoch - 6ms/step
Epoch 95/100
9/9 - 0s - loss: 0.1281 - accuracy: 0.8231 - val_loss: 0.1631 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 96/100
9/9 - 0s - loss: 0.1276 - accuracy: 0.8231 - val_loss: 0.1629 -
val_accuracy: 0.7500 - 55ms/epoch - 6ms/step
Epoch 97/100
9/9 - 0s - loss: 0.1272 - accuracy: 0.8231 - val_loss: 0.1625 -
val_accuracy: 0.7500 - 59ms/epoch - 7ms/step
Epoch 98/100
9/9 - 0s - loss: 0.1268 - accuracy: 0.8154 - val_loss: 0.1623 -
val_accuracy: 0.7500 - 56ms/epoch - 6ms/step
```

```
Epoch 99/100
9/9 - 0s - loss: 0.1264 - accuracy: 0.8154 - val_loss: 0.1617 -
val_accuracy: 0.7500 - 61ms/epoch - 7ms/step
Epoch 100/100
9/9 - 0s - loss: 0.1260 - accuracy: 0.8154 - val_loss: 0.1612 -
val_accuracy: 0.7500 - 51ms/epoch - 6ms/step
```

```
# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('LSTM Model Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




```
# Make predictions on test data
y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score

# Make predictions on the test set
y_pred = model.predict(X_test)

# Convert probabilities to classes
y_pred_classes = np.argmax(y_pred, axis=1)

# Compute accuracy
```

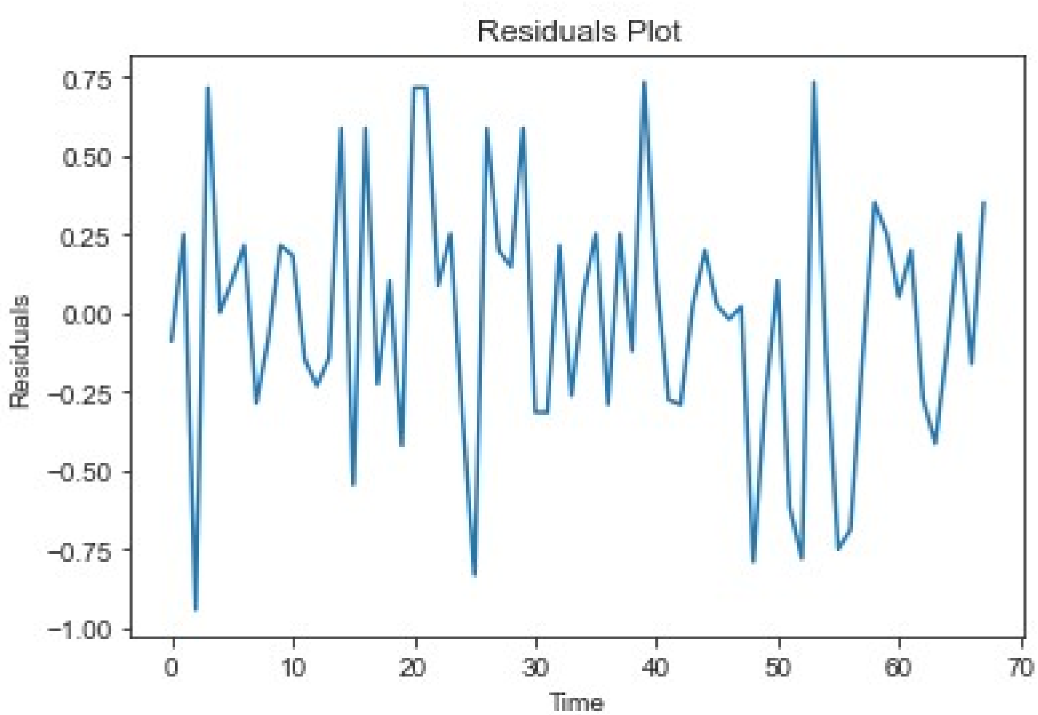
```
accuracy = accuracy_score(y_test, y_pred_classes)

print('Accuracy: {:.2f}'.format(accuracy))
```

```
Accuracy: 0.51
```

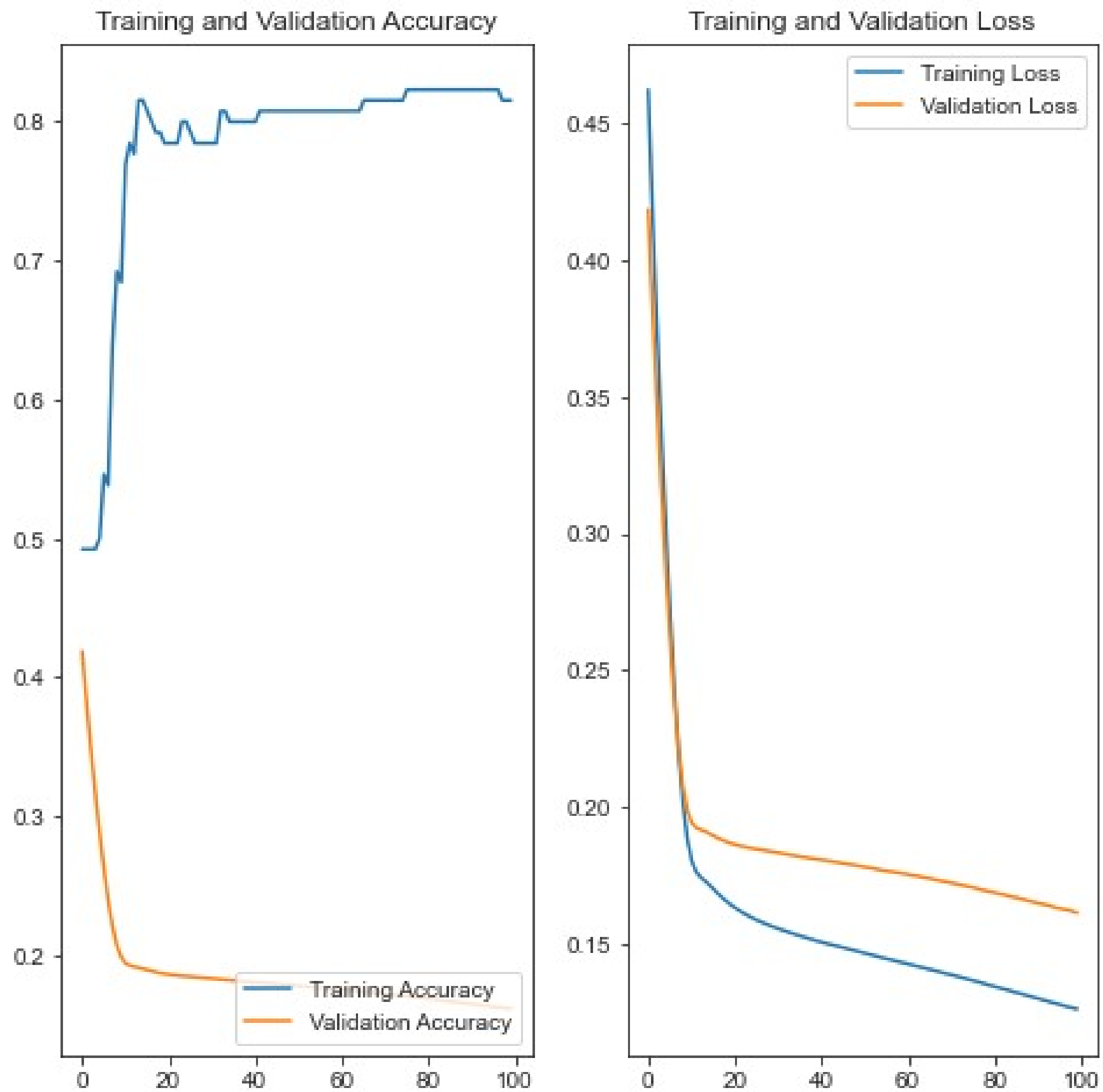
```
# Evaluate model performance using metrics as rmse, mae, mse, r2
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print('RMSE: {:.2f}'.format(rmse))
print('MAE: {:.2f}'.format(mae))
print('MSE: {:.2f}'.format(mse))
print('R2: {:.2f}'.format(r2))
```

```
RMSE: 0.40
MAE: 0.32
MSE: 0.16
R2: 0.35
```

```
# Plot residuals
residuals = y_test.ravel() - y_pred.ravel()
plt.plot(residuals)
plt.xlabel('Time')
plt.ylabel('Residuals')
plt.title('Residuals Plot')
plt.show()
```



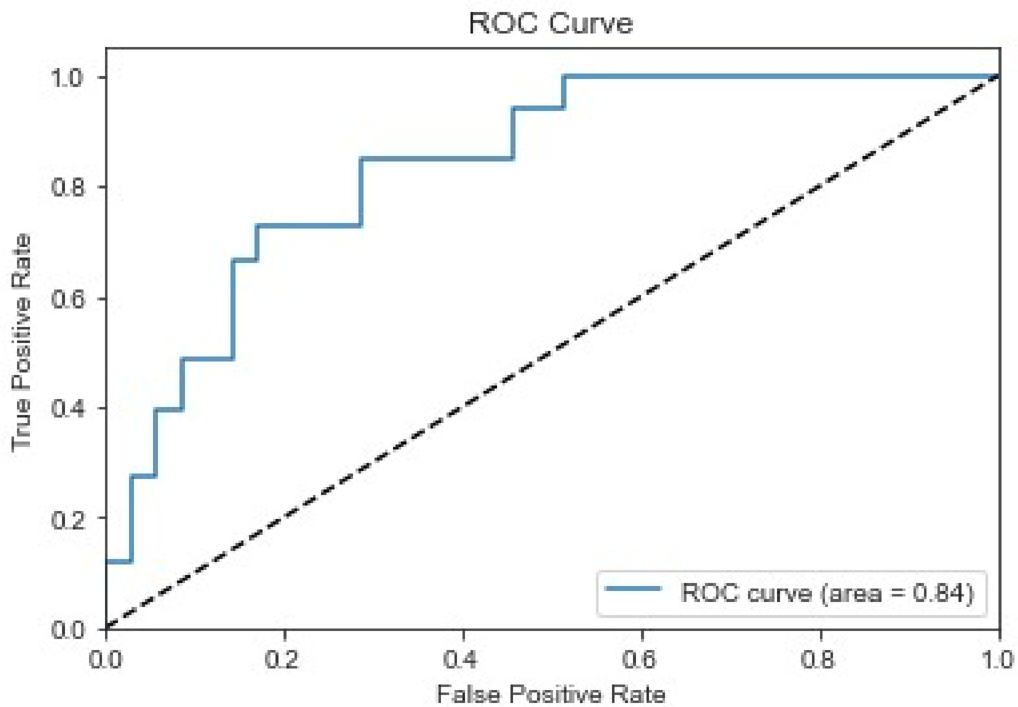
```python
#defining acc, val_acc, loss, val_loss, epochs and epochs_range
epochs=100
epochs_range = range(epochs)
acc = history.history['accuracy']
val_acc = history.history['val_loss']

loss= history.history['loss']
val_loss = history.history['val_loss']

#plot of learning process using accuracy, validation accuracy, loss
and validation loss information from model training

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range,
         acc,
         label='Training Accuracy')

plt.plot(epochs_range,
         val_acc,
         label='Validation Accuracy')

plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range,
         loss,
         label='Training Loss')

plt.plot(epochs_range,
         val_loss,
         label='Validation Loss')

plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



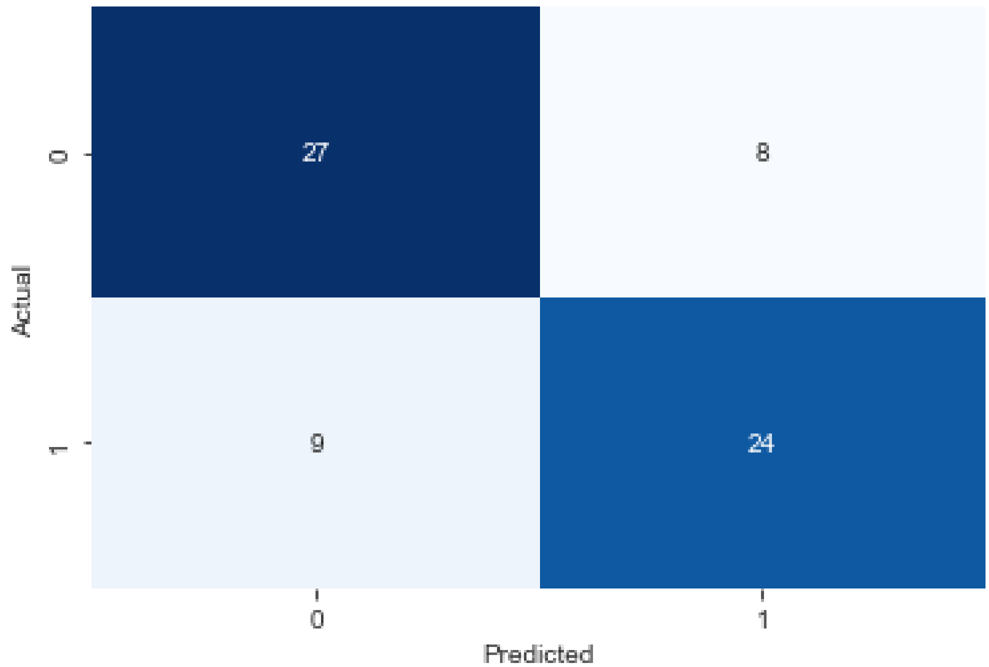

```python
from sklearn.metrics import roc_curve, auc
# Compute the false positive rate, true positive rate and threshold
for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)

# Compute the area under the ROC curve
roc_auc = auc(fpr, tpr)

print ("AUC-ROC score:", roc_auc)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC curve (area = {:.2f})'.format(roc_auc))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

```
AUC-ROC score: 0.8372294372294372
```




```
from sklearn.metrics import confusion_matrix
import seaborn as sns

# Get predictions on test set
y_pred = model.predict(X_test)

# Round predictions to 0 or 1
y_pred = np.round(y_pred)

# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix using seaborn heatmap
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```




```
#printing classification report from the y_test values and predictions

import warnings
from sklearn.exceptions import UndefinedMetricWarning
#from sklearn.metrics import precision_score, recall_score, f1_score

# Ignore UndefinedMetricWarning
warnings.filterwarnings('ignore', category=UndefinedMetricWarning)


print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.75      0.77      0.76        35
           1       0.75      0.73      0.74        33

    accuracy                           0.75        68
   macro avg       0.75      0.75      0.75        68
weighted avg       0.75      0.75      0.75        68
```