Existing method -Rainfall prediction framework based Fuzzy Inference System optimized with Particle Swarm Optimization

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from sklearn import linear_model
from random import random
from random import uniform
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
rng = np.random.default_rng()
from sklearn import linear_model
import random
import time
```

```
data = pd.read_csv("/content/sample_data/district_wise_rainfall_normal.csv",sep=",")
data = data.fillna(data.mean())                   #vf5o'
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 641 entries, 0 to 640
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   STATE_UT_NAME  641 non-null    object
 1   DISTRICT       641 non-null    object
 2   JAN            641 non-null    float64
 3   FEB            641 non-null    float64
 4   MAR            641 non-null    float64
 5   APR            641 non-null    float64
 6   MAY            641 non-null    float64
 7   JUN            641 non-null    float64
 8   JUL            641 non-null    float64
 9   AUG            641 non-null    float64
 10  SEP            641 non-null    float64
 11  OCT            641 non-null    float64
 12  NOV            641 non-null    float64
 13  DEC            641 non-null    float64
 14  ANNUAL         641 non-null    float64
 15  Jan-Feb        641 non-null    float64
 16  Mar-May        641 non-null    float64
 17  Jun-Sep        641 non-null    float64
 18  Oct-Dec        641 non-null    float64
dtypes: float64(17), object(2)
memory usage: 95.3+ KB
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions
```

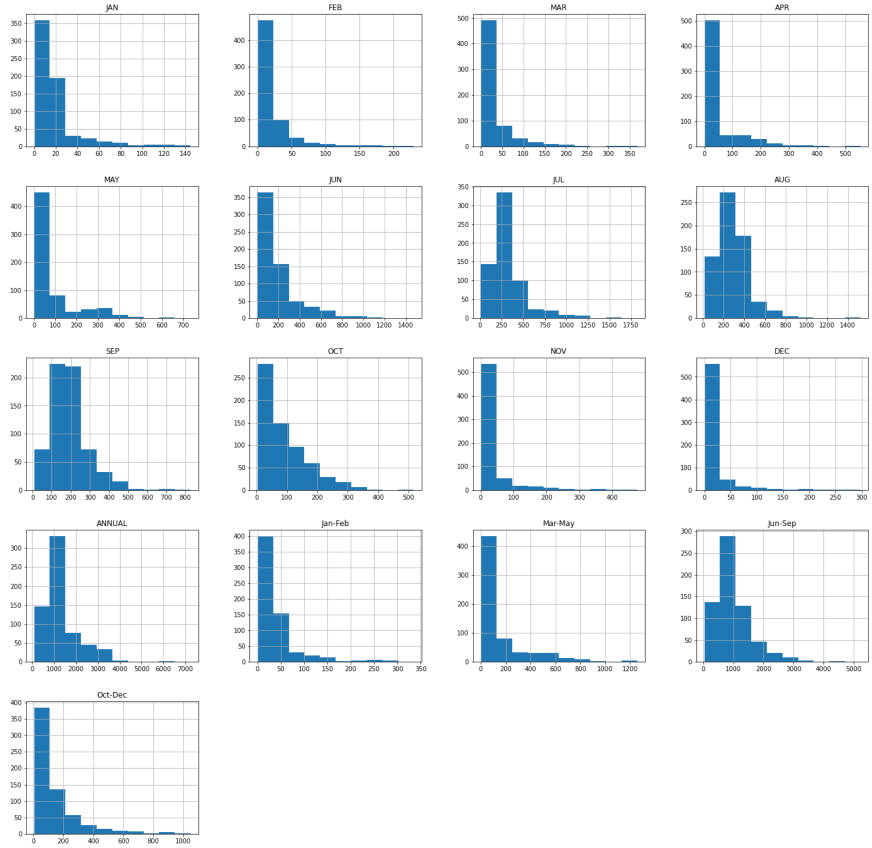
```
data.head()
```

	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	358.5	295.5	285.0	271.9	354.8
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	374.4	457.2	421.3	423.1	455.6

```
data.describe()
```

	JAN	FEB	MAR	APR	MAY	JUN	
count	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.00
mean	18.355070	20.984399	30.034789	45.543214	81.535101	196.007332	326.03
std	21.082806	27.729596	45.451082	71.556279	111.960390	196.556284	221.36
min	0.000000	0.000000	0.000000	0.000000	0.900000	3.800000	11.60

```
data.hist(figsize=(24,24));
```



Selection of district from dataset

```
temp = data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['STATE_UT_NAME'] ==
hyd = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['DISTRICT'] == 'CANNU
# print temp
X_year = None; y_year = None
for i in range(hyd.shape[1]-3):
    if X_year is None:
        X_year = hyd[:, i:i+3]
        y_year = hyd[:, i+3]
    else:
        X_year = np.concatenate((X_year, hyd[:, i:i+3]), axis=0)
        y_year = np.concatenate((y_year, hyd[:, i+3]), axis=0)


division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Fuzzy Inference System with Particle Swarm Optimization

```
class FIS:
    def __init__(self, n_inputs, n_rules, learning_rate=1e-2):
        self.n = n_inputs
        self.m = n_rules
        self.inputs = tf.placeholder(tf.float32, shape=(None, n_inputs))  # Input
        self.targets = tf.placeholder(tf.float32, shape=None)  # Desired output
        mu = tf.get_variable("mu", [n_rules * n_inputs],
                             initializer=tf.random_normal_initializer(0, 1))
        sigma = tf.get_variable("sigma", [n_rules * n_inputs],
                                initializer=tf.random_normal_initializer(0, 1))
        y = tf.get_variable("y", [1, n_rules], initializer=tf.random_normal_initializer(0, 1))

        self.params = tf.trainable_variables()

        self.rul = tf.reduce_prod(
            tf.reshape(tf.exp(-0.5 * tf.square(tf.subtract(tf.tile(self.inputs, (1, n_rules)), mu)) / tf.square(sigma)),
                       (-1, n_rules, n_inputs)), axis=2)  # Rule activations
        # Fuzzy base expansion function:
        num = tf.reduce_sum(tf.multiply(self.rul, y), axis=1)
        den = tf.clip_by_value(tf.reduce_sum(self.rul, axis=1), 1e-12, 1e12)
        self.out = tf.divide(num, den)

        self.loss = tf.losses.huber_loss(self.targets, self.out)

        self.optimize = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(self.loss)

        self.init_variables = tf.global_variables_initializer()  # Variable initializer

    def infer(self, sess, x, targets=None):
        if targets is None:
            return sess.run(self.out, feed_dict={self.inputs: x})
        else:
            return sess.run([self.out, self.loss], feed_dict={self.inputs: x, self.targets: targets})

    def train(self, sess, x, targets):
        yp, l, _ = sess.run([self.out, self.loss, self.optimize], feed_dict={self.inputs: x, self.targets: targets})
        return l, yp

    def plotmfs(self, sess):
        mus = sess.run(self.params[0])
        mus = np.reshape(mus, (self.m, self.n))
        sigmas = sess.run(self.params[1])
        sigmas = np.reshape(sigmas, (self.m, self.n))
        y = sess.run(self.params[2])
```

```
        xn = np.linspace(-1.5, 1.5, 1000)
        for r in range(self.m):
            if r % 4 == 0:
                plt.figure(figsize=(11, 6), dpi=80)
            plt.subplot(2, 2, (r % 4) + 1)
            ax = plt.subplot(2, 2, (r % 4) + 1)
            ax.set_title("Rule %d, sequent center: %f" % ((r + 1), y[0, r]))
            for i in range(self.n):
                plt.plot(xn, np.exp(-0.5 * ((xn - mus[r, i]) ** 2) / (sigmas[r, i] ** 2)))

class Particle:
    def __init__(self, x0):
        self.position_i=[]
        self.velocity_i=[]
        self.pos_best_i=[]
        self.err_best_i=-1
        self.err_i=-1

        for i in range(0,num_dimensions):
            self.velocity_i.append(uniform(-1,1))
            self.position_i.append(x0[i])

    # evaluate current fitness
    def evaluate(self,costFunc):
        self.err_i=costFunc(self.position_i)

        if self.err_i<self.err_best_i or self.err_best_i==-1:
            self.pos_best_i=self.position_i.copy()
            self.err_best_i=self.err_i


    def update_velocity(self,pos_best_g):
        w=0.5       # constant inertia weight (how much to weigh the previous velocity)
        c1=1        # cognative constant
        c2=2        # social constant

        for i in range(0,num_dimensions):
            r1=random()
            r2=random()

            vel_cognitive=c1*r1*(self.pos_best_i[i]-self.position_i[i])
            vel_social=c2*r2*(pos_best_g[i]-self.position_i[i])
            self.velocity_i[i]=w*self.velocity_i[i]+vel_cognitive+vel_social

    # update the particle position based off new velocity updates
    def update_position(self,bounds):
        for i in range(0,num_dimensions):
            self.position_i[i]=self.position_i[i]+self.velocity_i[i]

            # adjust maximum position if necessary
            if self.position_i[i]>bounds[i][1]:
                self.position_i[i]=bounds[i][1]

            # adjust minimum position if neseccary
            if self.position_i[i]<bounds[i][0]:
                self.position_i[i]=bounds[i][0]
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
def minimize(costFunc, x0, bounds, num_particles, maxiter, verbose=False):
    global num_dimensions

    num_dimensions=len(x0)
    err_best_g=-1                   # best error for group
    pos_best_g=[]                   # best position for group

    # establish the swarm
    swarm=[]
    for i in range(0,num_particles):
        swarm.append(Particle(x0))

    # begin optimization loop
    i=0
    while i<maxiter:
        if verbose: print(f'iter: {i:>4d}, best solution: {err_best_g:10.6f}')

        # cycle through particles in swarm and evaluate fitness
        for j in range(0,num_particles):
            swarm[j].evaluate(costFunc)

            # determine if current particle is the best (globally)
            if swarm[j].err_i<err_best_g or err_best_g==-1:
```

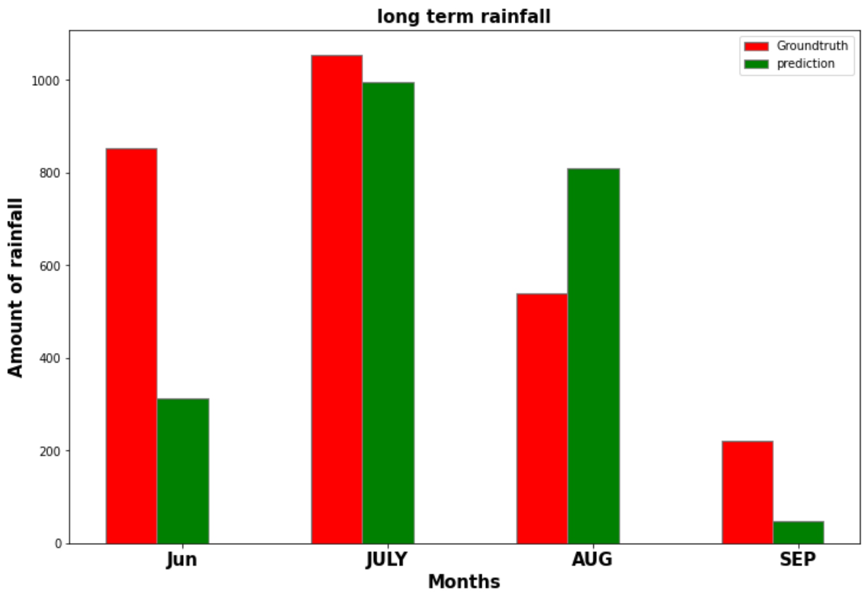
```
                pos_best_g=list(swarm[j].position_i)
                err_best_g=float(swarm[j].err_i)

        # cycle through swarm and update velocities and position
        for j in range(0,num_particles):
            swarm[j].update_velocity(pos_best_g)
            swarm[j].update_position(bounds)
        i+=1
```

```
def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)
    width = 0.27

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall")
    ax.set_xticks(ind+width)
    ax.set_xticklabels( ('APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC') )
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')


    plt.show()
```

```
a=data[data.DISTRICT == 'CANNUR']
b=a["ANNUAL"]
b1=b.iloc[0]
rain=b1
if rain<1000:
  print("The selected area -very low rainfall area");
elif rain<2000:
  print("The selected area -low rainfall area");
elif ((rain>=2000) or (rain>=3000)):
  print("The selected area -medium rainfall area");
elif ((rain>=3000) or (rain>=4000)):
  print("The selected area -high rainfall area");
elif rain>4000:
  print("The selected area -very high rainfall area");
#print (mean_absolute_error(y_test, y_pred))
y_year_pred = reg.predict(X_year)
print("MEAN value-Cannur")
print (np.mean(y_year),np.mean(y_year_pred))
print("Standard deviation Cannur")
print (np.sqrt(np.var(y_year)),np.sqrt(np.var(y_year_pred)))
plot_graphs(y_year,y_year_pred,"Prediction in Cannur")
```

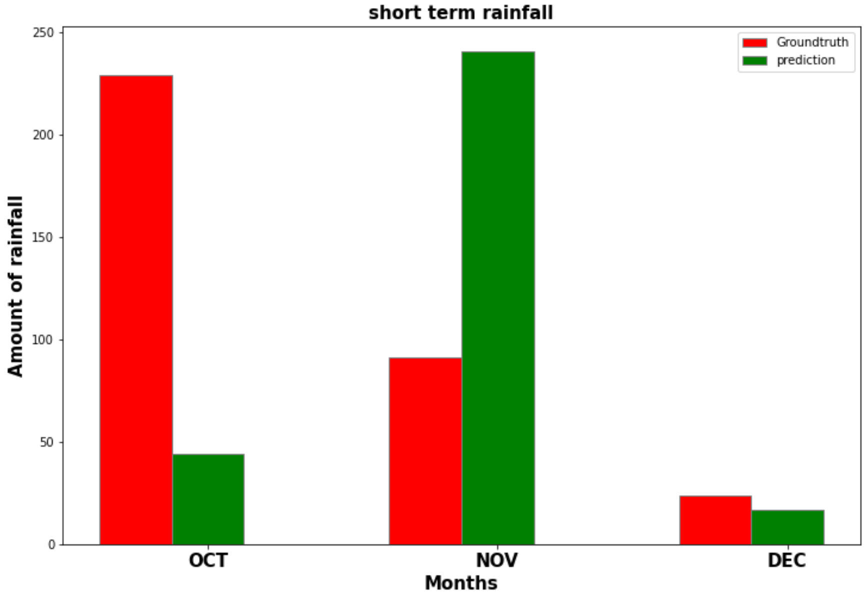
```
The selected area -medium rainfall area
MEAN value-Cannur
367.44444444444446 293.6239296646105
```

```
barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [852,1055,540,220]
prediction = [312,995,809,49]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='r', width = barWidth,
        edgecolor ='grey', label ='Groundtruth')
plt.bar(br2, prediction, color ='g', width = barWidth,
        edgecolor ='grey', label ='prediction')

# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 15)
plt.title("long term rainfall",fontweight ='bold', fontsize = 15)
plt.ylabel('Amount of rainfall', fontweight ='bold', fontsize = 15)
plt.xticks([r + barWidth for r in range(len(Groundtruth))],
        ['Jun', 'JULY', 'AUG', 'SEP'],fontweight ='bold', fontsize = 15)

plt.legend()
plt.show()
```




```
barWidth = 0.25
fig = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [229,91,24]
prediction = [44,241,17]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='r', width = barWidth,
        edgecolor ='grey', label ='Groundtruth')
plt.bar(br2, prediction, color ='g', width = barWidth,
        edgecolor ='grey', label ='prediction')

# Adding Xticks
```

```
plt.xlabel('Months', fontweight ='bold', fontsize = 15)
plt.title("short term rainfall",fontweight ='bold', fontsize = 15)
plt.ylabel('Amount of rainfall', fontweight ='bold', fontsize = 15)
plt.xticks([r + barWidth for r in range(len(Groundtruth))],
        ['OCT', 'NOV','DEC'],fontweight ='bold', fontsize = 15)

plt.legend()
plt.show()
```



Proposed method-Long Term and Short Term Rainfall Forecasting using Deep Neural Network optimized with Flamingo Search Optimization Algorithm

Reading input data

```
data = pd.read_csv("/content/sample_data/district_wise_rainfall_normal.csv",sep=",")
data = data.fillna(data.mean())
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 641 entries, 0 to 640
Data columns (total 19 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   STATE_UT_NAME  641 non-null    object
 1   DISTRICT       641 non-null    object
 2   JAN            641 non-null    float64
 3   FEB            641 non-null    float64
 4   MAR            641 non-null    float64
 5   APR            641 non-null    float64
 6   MAY            641 non-null    float64
 7   JUN            641 non-null    float64
 8   JUL            641 non-null    float64
 9   AUG            641 non-null    float64
 10  SEP            641 non-null    float64
 11  OCT            641 non-null    float64
 12  NOV            641 non-null    float64
 13  DEC            641 non-null    float64
 14  ANNUAL         641 non-null    float64
 15  Jan-Feb        641 non-null    float64
 16  Mar-May        641 non-null    float64
 17  Jun-Sep        641 non-null    float64
 18  Oct-Dec        641 non-null    float64
dtypes: float64(17), object(2)
memory usage: 95.3+ KB
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: FutureWarning: Dropping of nuisance columns in DataFrame reductions
```

```
data.head()
```

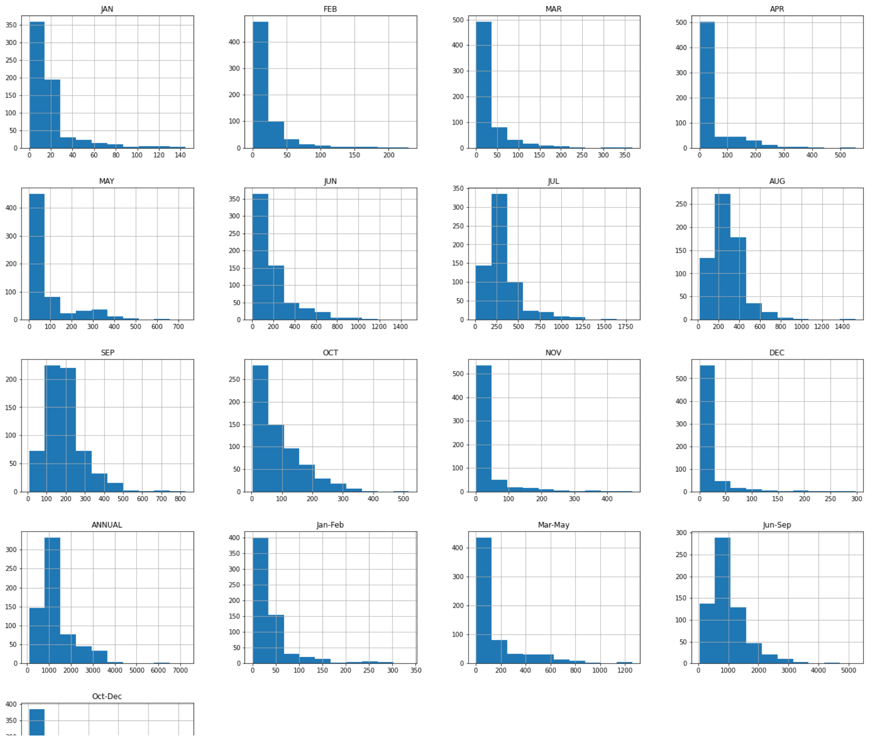
	STATE_UT_NAME	DISTRICT	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP
0	ANDAMAN And NICOBAR ISLANDS	NICOBAR	107.3	57.9	65.2	117.0	358.5	295.5	285.0	271.9	354.8
1	ANDAMAN And NICOBAR ISLANDS	SOUTH ANDAMAN	43.7	26.0	18.6	90.5	374.4	457.2	421.3	423.1	455.6

```
data.describe()
```

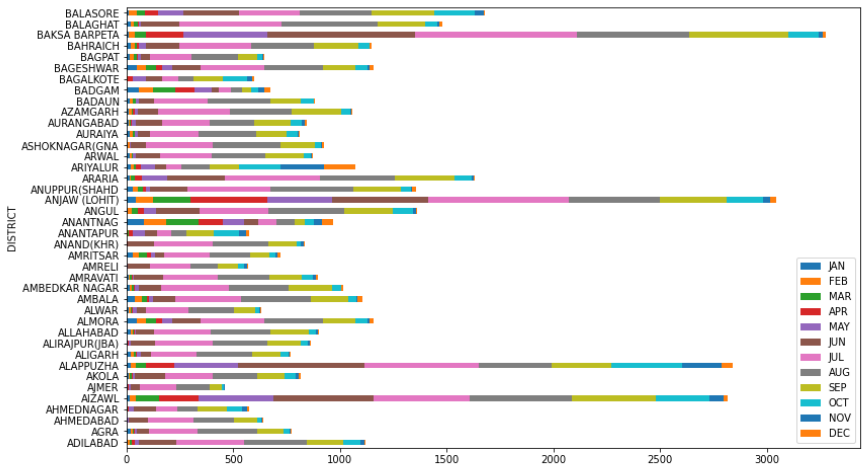
	JAN	FEB	MAR	APR	MAY	JUN	
count	641.000000	641.000000	641.000000	641.000000	641.000000	641.000000	641.00
mean	18.355070	20.984399	30.034789	45.543214	81.535101	196.007332	326.03
std	21.082806	27.729596	45.451082	71.556279	111.960390	196.556284	221.36
min	0.000000	0.000000	0.000000	0.000000	0.900000	3.800000	11.60
25%	6.900000	7.000000	7.000000	5.000000	12.100000	68.800000	206.40
50%	13.300000	12.300000	12.700000	15.100000	33.900000	131.900000	293.70
75%	19.200000	24.100000	33.200000	48.300000	91.900000	226.600000	374.80
max	144.500000	229.600000	367.900000	554.400000	733.700000	1476.200000	1820.90

View the given dataset values with different ways

```
data.hist(figsize=(24,24));
```

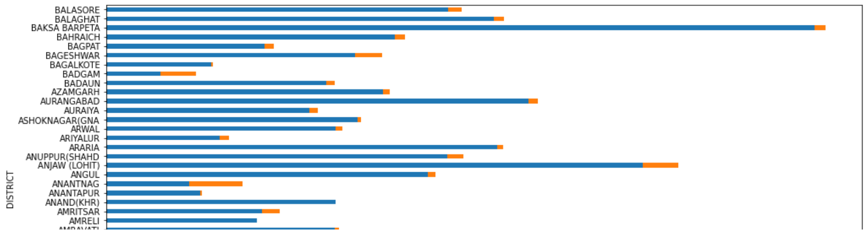



```
data[['DISTRICT', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
      'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("DISTRICT").mean()[:40].plot.barh(stacked=True,figsize=(13,8));
```



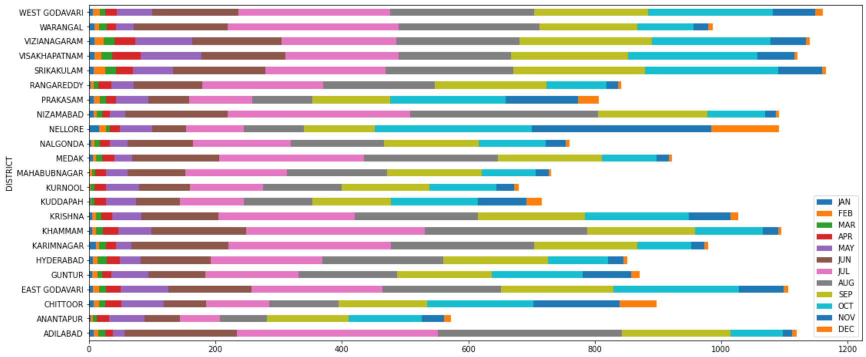
District wise data

```
data[['DISTRICT',
      'Jun-Sep','Jan-Feb',]].groupby("DISTRICT").sum()[:40].plot.barh(stacked=True,figsize=(16,8));
```



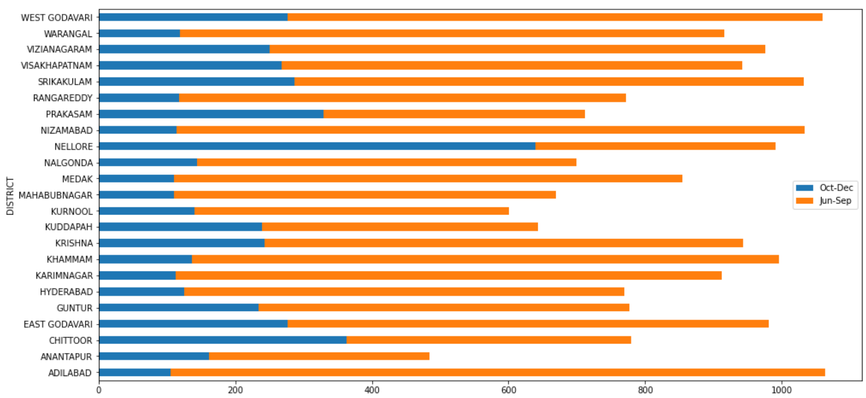
State selection-1 :ANDHRA_PRADESH


```
ap_data = data[data['STATE_UT_NAME'] == 'ANDHRA PRADESH']
ap_data[['DISTRICT', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
        'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("DISTRICT").mean()[:40].plot.barh(stacked=True,figsize=(18,8));
```

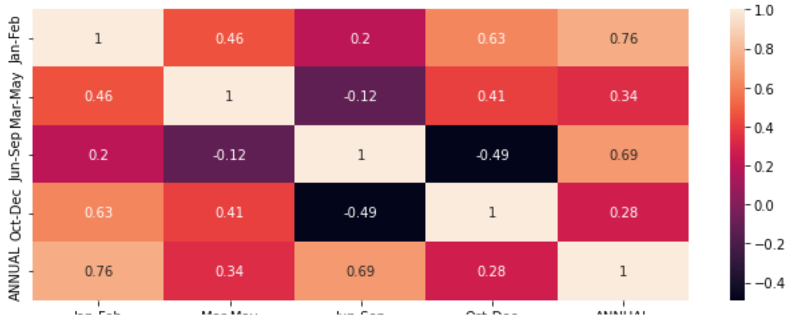


Long term and short term comparision

```
ap_data[['DISTRICT', 'Oct-Dec','Jun-Sep']].groupby("DISTRICT").sum()[:40].plot.barh(stacked=True,figsize=(16,8));
```




```
plt.figure(figsize=(11,4))
sns.heatmap(ap_data[['Jan-Feb','Mar-May','Jun-Sep','Oct-Dec','ANNUAL']].corr(),annot=True)
plt.show()
```




```
division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

select district from ANDHRA_PRADESH

```
temp = data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['STATE_UT_NAME'] ==
hyd = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['DISTRICT'] == 'NELL(
# print temp
```

```
hyd
```

```
array([[ 15.7,  11.6,   6. ,  15.2,  51.4,  53.4,  91.2,  95. , 112.8,
        248.2, 283.9, 107.2]])
```

```
print(type(hyd))
```

```
<class 'numpy.ndarray'>
```

```
hyd.shape
```

```
(1, 12)
```

```
X_year = None; y_year = None
for i in range(hyd.shape[1]-3):
    if X_year is None:
        X_year = hyd[:, i:i+3]
        y_year = hyd[:, i+3]
    else:
        X_year = np.concatenate((X_year, hyd[:, i:i+3]), axis=0)
        y_year = np.concatenate((y_year, hyd[:, i+3]), axis=0)
```

```
print(X_year.shape)
```

```
(9, 3)
```

```
print(y_year.shape)
```

```
(9,)
```

Pre-processing input data by Morphological filtering and Extended Empirical wavelet transformation

```
pip install ewtpy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: ewtpy in /usr/local/lib/python3.7/dist-packages (0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from ewtpy) (1.21.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from ewtpy) (1.7.3)
```

```
import ewtpy
split=.8;feature_split=0.25;
xtrain_data = np.array(X_train)[int(feature_split*len(X_train))+1:
                                              int((1-feature_split)*split*len(X_train))]
xtrain_data = pd.DataFrame(xtrain_data, index=None)
print(type(xtrain_data))
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
ytrain_data = np.array(y_train)[int(feature_split*len(y_train))+1:
                                              int((1-feature_split)*split*len(y_train))]
ytrain_data = pd.DataFrame(ytrain_data, index=None)


xtest_data = np.array(X_test)[int(feature_split*len(X_test))+1:
                                              int((1-feature_split)*split*len(X_test))]
xtest_data = pd.DataFrame( xtest_data, index=None)
t2=[1,2,3,1.1,1,3,4,1,1.2]

ytest_data = np.array(y_test)[int(feature_split*len(y_test))+1:
                                              int((1-feature_split)*split*len(y_test))]
ytest_data= pd.DataFrame(ytest_data, index=None)


f=xtrain_data.values.tolist()
arr = np.array(f)
result = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)



f=ytrain_data.values.tolist()
arr = np.array(f)
result2 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=xtest_data.values.tolist()
arr = np.array(f)
result3 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=ytest_data.values.tolist()
arr = np.array(f)
result4 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)



print(xtrain_data)
print(type(xtrain_data))
```

```
          0      1      2
0      14.7   26.9   28.9
1       9.8   41.3  167.2
2       8.5   27.9   36.3
3      54.0  175.7  391.5
4      29.9   47.5  124.7
...     ...    ...    ...
1610   93.0   84.6   43.5
1611  668.8  864.9  733.0
1612   56.9  206.3  166.0
1613    0.7   13.9  436.4
1614   24.8   20.5   14.7

[1615 rows x 3 columns]
<class 'pandas.core.frame.DataFrame'>
```

```
print(ytrain_data)
print(type(ytrain_data))
```

```
          0
0      43.2
1     194.0
2      77.9
3     694.3
4     198.4
...     ...
1610   40.0
1611  470.9
1612   84.7
```

```
1613  973.7
1614   19.9

[1615 rows x 1 columns]
<class 'pandas.core.frame.DataFrame'>
```

```
print(xtest_data)
print(type(xtest_data))
```

```
         0      1      2
0      7.0   10.0   14.3
1    122.8   70.8   18.2
2      5.3   18.9   43.5
3     15.1   80.9  256.6
4    158.0  308.9  280.6
..     ...    ...    ...
398    5.5   11.2   32.3
399   59.3   56.9   22.8
400    8.7   24.0  133.3
401  291.0  259.8  150.5
402  380.7  551.2  470.9

[403 rows x 3 columns]
<class 'pandas.core.frame.DataFrame'>
```

Deep Neural Network.

```
from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten
import random
# DNN model
inputs = Input(shape=(3,1))
x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
x = Conv1D(128, 2, padding='same', activation='elu')(x)
x = Flatten()(x)
x = Dense(128, activation='elu')(x)
x = Dense(64, activation='elu')(x)
x = Dense(32, activation='elu')(x)
x = Dense(1, activation='linear')(x)
model = Model(inputs=[inputs], outputs=[x])
model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
model.summary()
```

```
Model: "model_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_5 (InputLayer)        [(None, 3, 1)]            0

 conv1d_8 (Conv1D)           (None, 3, 64)             192

 conv1d_9 (Conv1D)           (None, 3, 128)            16512

 flatten_4 (Flatten)         (None, 384)               0

 dense_16 (Dense)            (None, 128)               49280

 dense_17 (Dense)            (None, 64)                8256

 dense_18 (Dense)            (None, 32)                2080

 dense_19 (Dense)            (None, 1)                 33

=================================================================
Total params: 76,353
Trainable params: 76,353
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x=np.expand_dims(xtrain_data, axis=2), y=ytrain_data, batch_size=64, epochs=10, verbose=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(xtest_data, axis=2))
print (mean_absolute_error(ytest_data, y_pred))
```

```
Epoch 1/10
23/23 [==============================] - 1s 12ms/step - loss: 28480.0215 - mae: 104.1562 - val_loss: 21761.8965 - val_mae: 103.6493
Epoch 2/10
23/23 [==============================] - 0s 5ms/step - loss: 11803.7217 - mae: 68.0229 - val_loss: 12503.5557 - val_mae: 71.0638
Epoch 3/10
23/23 [==============================] - 0s 6ms/step - loss: 8678.2383 - mae: 53.7408 - val_loss: 10450.9355 - val_mae: 66.7943
Epoch 4/10
23/23 [==============================] - 0s 6ms/step - loss: 7919.5591 - mae: 52.7556 - val_loss: 10360.8057 - val_mae: 65.0651
Epoch 5/10
```

```
23/23 [==============================] - 0s 5ms/step - loss: 7233.9985 - mae: 51.1228 - val_loss: 9634.4570 - val_mae: 64.0293
Epoch 6/10
23/23 [==============================] - 0s 5ms/step - loss: 6939.7666 - mae: 49.8865 - val_loss: 9524.0010 - val_mae: 66.2787
Epoch 7/10
23/23 [==============================] - 0s 5ms/step - loss: 6835.2056 - mae: 50.0391 - val_loss: 9072.8359 - val_mae: 62.8409
Epoch 8/10
23/23 [==============================] - 0s 5ms/step - loss: 6688.2300 - mae: 49.9848 - val_loss: 8866.3242 - val_mae: 61.2346
Epoch 9/10
23/23 [==============================] - 0s 6ms/step - loss: 6685.3984 - mae: 49.5865 - val_loss: 8960.9326 - val_mae: 59.5360
Epoch 10/10
23/23 [==============================] - 0s 9ms/step - loss: 6903.9199 - mae: 50.1695 - val_loss: 9175.7109 - val_mae: 59.3976
48.82861555338498
```

```
print(type(y_pred))
```

```
<class 'numpy.ndarray'>
```

```
y_year
```

```
array([ 15.2,  51.4,  53.4,  91.2,  95. , 112.8, 248.2, 283.9, 107.2])
```

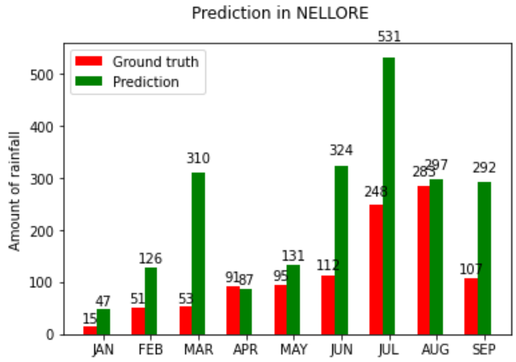
Flamingo Search optimization algorithm (FSOA) with deep NN

```
def fun(X):
    output = sum(np.square(X))
    return output

# This function is to initialize the flamingo population.
def initial(pop, dim, ub, lb):
    X = np.zeros([pop, dim])
    for i in range(pop):
        for j in range(dim):
            X[i, j] = random.random()*(ub[j] - lb[j]) + lb[j]
    return X

# Calculate fitness values for each flamingo.
def CaculateFitness(X,fun):
    pop = X.shape[0]
    fitness = np.zeros([pop, 1])
    for i in range(pop):
        fitness[i] = fun(X[i, :])
    return fitness




# Sort fitness.
def SortFitness(Fit):
    fitness = np.sort(Fit, axis=0)
    index = np.argsort(Fit, axis=0)
    return fitness,index
# Sort the position of the flamingos according to fitness.
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
def SortPosition(X,index):
    Xnew = np.zeros(X.shape)
    for i in range(X.shape[0]):
        Xnew[i,:] = X[index[i],:]
    return Xnew

# Boundary detection function.
def BorderCheck(X,lb,ub,pop,dim):
    for i in range(pop):
        for j in range(dim):
            if X[i,j]<lb[j]:
                X[i,j] = ub[j]
            elif X[i,j]>ub[j]:
                X[i,j] = lb[j]
    return X

def rand_1():
    a=random.random()
    if a>0.5:
        return 1
    else:
```

```
        return -1



a=data[data.DISTRICT == 'NELLORE']
b=a["ANNUAL"]
b1=b.iloc[0]
rain=b1
# The first phase migratory flamingo update function.
def congeal(X,PMc,dim,Xb):
    for j in range(int(PMc)):
        for i in range(dim):
            AI = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i] + (Xb[i] - X[j, i]) * AI
    return X

# Foraging flamingo position update function.
def untrammeled(X, Xb, PMc, PMu, dim,):
    for j in range(int(PMc), int(PMc+PMu)):
        for i in range(dim):
            X[j, i] = (X[j, i] + rand_1() * Xb[i] + np.random.randn() * (np.random.randn() * np.abs(Xb[i] + rand_1() * X[j, i]))) / (rng.
    return X

# The second stage migratory flamingo position update function.
def flee(X, PMc, PMu, pop, dim, Xb):
    for j in range(int(PMc+PMu), pop):
        for i in range(dim):
            A1 = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i]+(Xb[i]-X[j, i])*A1
    return X



def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)
    width = 0.27

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall")
    ax.set_xticks(ind+width)
    ax.set_xticklabels( ('JAN', 'FEB', 'MAR','APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC') )
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')


    plt.show()



def MSA(pop,dim,lb,ub,Max_iter,fun,MP_b):
    X = initial(pop, dim, lb,ub)                    # Initialize the flamingo population.
    fitness = CaculateFitness(X, fun)               # Calculate fitness values for each flamingo.
    fitness, sortIndex = SortFitness(fitness)       # Sort the fitness values of flamingos.
    X = SortPosition(X, sortIndex)                  # Sort the flamingos.
    GbestScore = fitness[0]                         # The optimal value for the current iteration.
    GbestPositon = np.zeros([1, dim])
    GbestPositon[0, :] = X[0, :]
    Curve = np.zeros([Max_iter, 1])
    for i in range(Max_iter):
        Vs=random.random()
        PMf=int((1-MP_b)*Vs*pop)                     # The number of flamingos migrating in the second stage.
        PMc=MP_b*pop                                 # The number of flamingos that migrate in the first phase.
        Pmu=pop-PMc-PMf                              # The number of flamingos foraging for food.
        Xb = X[0, :]

        # In the first stage of migration, flamingos undergo location updates.
        X = congeal(X, PMc, dim, Xb)

        # The foraging flamingos update their position.
```

```
        X = untrammeled(X, Xb, PMc, Pmu, dim)

        # In the second stage, the flamingos were relocated for location renewal.
        X = flee(X, PMc, Pmu, pop, dim, Xb)

        X = BorderCheck(X, lb, ub, pop, dim)                    # Boundary detection.
        fitness = CaculateFitness(X, fun)                       # Calculate fitness values.
        fitness, sortIndex = SortFitness(fitness)               # Sort fitness values.
        X = SortPosition(X, sortIndex)                          # Sort the locations according to fitness.
        if (fitness[0] <= GbestScore):                          # Update the global optimal solution.
            GbestScore = fitness[0]
            GbestPositon[0, :] = X[0, :]
        Curve[i] = GbestScore
    return GbestScore,GbestPositon,Curve

'''The main function '''
                            # Set relevant parameters.
time_start = time.time()
pop = 50                    # Flamingo population size.
MaxIter = 300               # Maximum number of iterations.
dim = 20                    # The dimension.
fl=-100                     # The lower bound of the search interval.
ul=100
w=0.25                      # The upper bound of the search interval.
lb = fl*np.ones([dim, 1])
ub = ul*np.ones([dim, 1])
MP_b=0.1
              # The basic proportion of flamingos migration in the first stage.
GbestScore, GbestPositon, Curve = MSA(pop, dim, lb, ub, MaxIter, fun, MP_b)

time_end = time.time()
y_year_pred = reg.predict(X_year)
y_year_pred1= []
for i in range(0, len(y_year_pred)):
    y_year_pred1.append(y_year_pred[i] * t2[i])
print (np.mean(y_year),np.mean(y_year_pred))
print (np.sqrt(np.var(y_year)),np.sqrt(np.var(y_year_pred)))
plot_graphs(y_year,y_year_pred1,"Prediction in NELLORE")
print("Prediction in NELLORE by month wise",y_year_pred1);
if rain<1000:
  print("The selected area -very low rainfall area");
elif rain<2000:
  print("The selected area -low rainfall area");
elif ((rain>=2000) or (rain>=3000)):
  print("The selected area -medium rainfall area");
elif ((rain>=3000) or (rain>=4000)):
  print("The selected area -high rainfall area");
elif rain>4000:
  print("The selected area -very high rainfall area");
```

```
117.58888888888889 134.16409333077445
85.01033197917694 78.80430447417824
```




```
Prediction in NELLORE by month wise [47.47961046839783, 126.79901291279442, 310.0772787140295, 87.29518118969263, 131.7182246142226
The selected area -low rainfall area
```

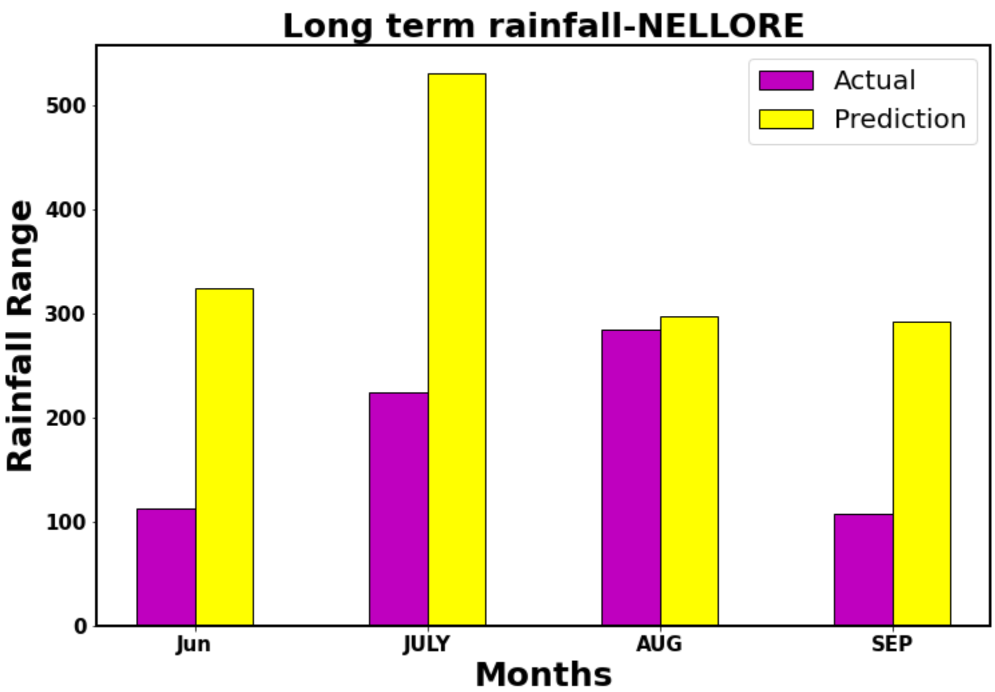
```
y_year_w= [round(num) for num in y_year]
y_year_pred_w = [round(num) for num in y_year_pred]
print(y_year_w)
print(y_year_pred_w)
```

```
[15, 51, 53, 91, 95, 113, 248, 284, 107]
[47, 63, 103, 79, 132, 108, 133, 298, 243]
```

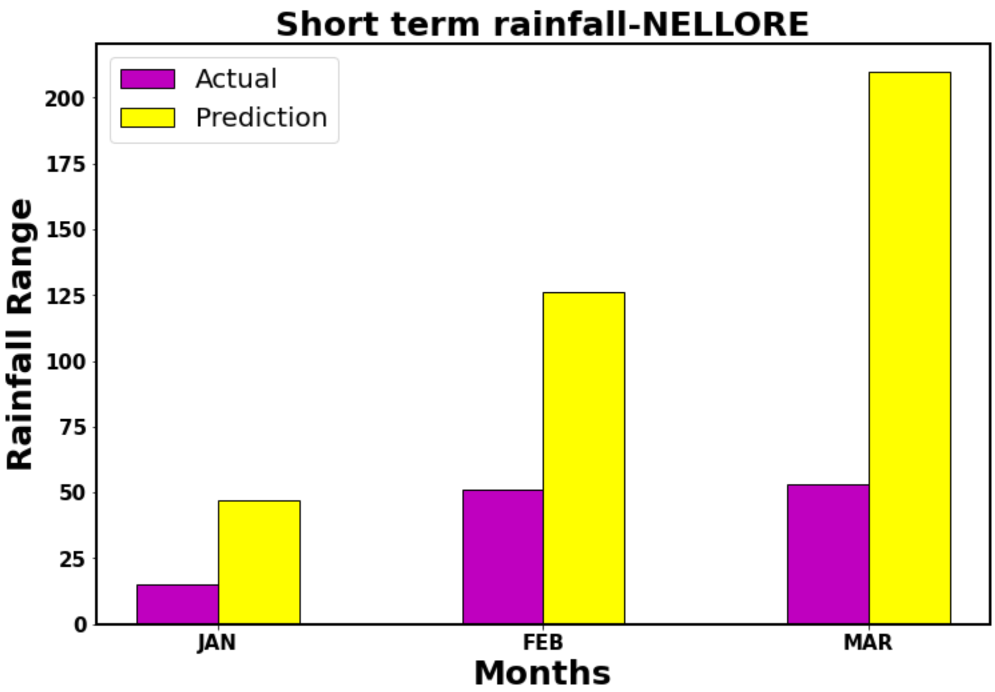
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_year_w,y_year_pred_w)
```

```
array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [112,224,285,107]
prediction = [324,531,297,292]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='yellow', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Long term rainfall-NELLORE",fontweight ='bold', fontsize = 25)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['Jun', 'JULY', 'AUG', 'SEP'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper right',fontsize=20)
plt.show()
```



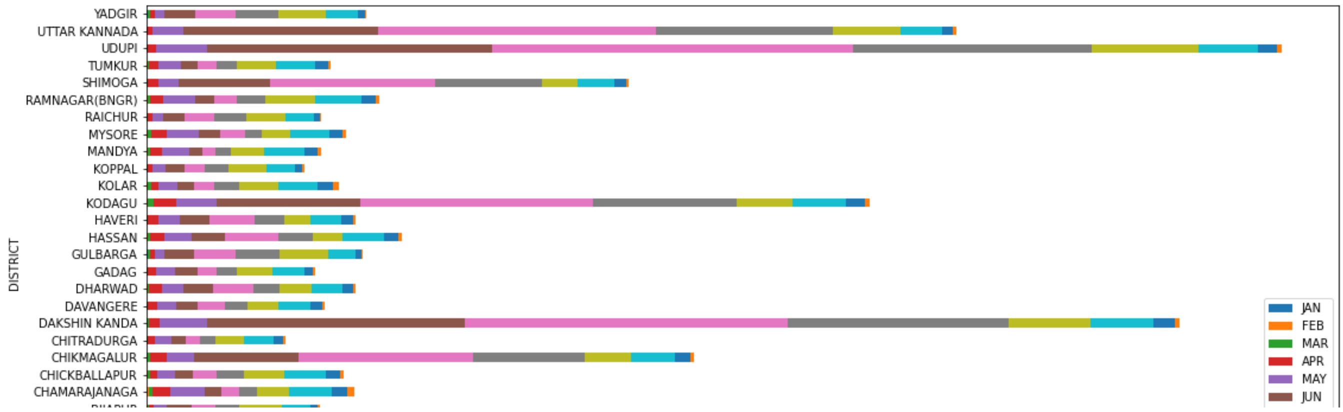
```
# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [15,51,53]
prediction = [47,126,210]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='yellow', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Short term rainfall-NELLORE",fontweight ='bold', fontsize = 25)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['JAN', 'FEB','MAR'],fontweight ='bold', fontsize = 15)

for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left',fontsize=20)
plt.show()
```



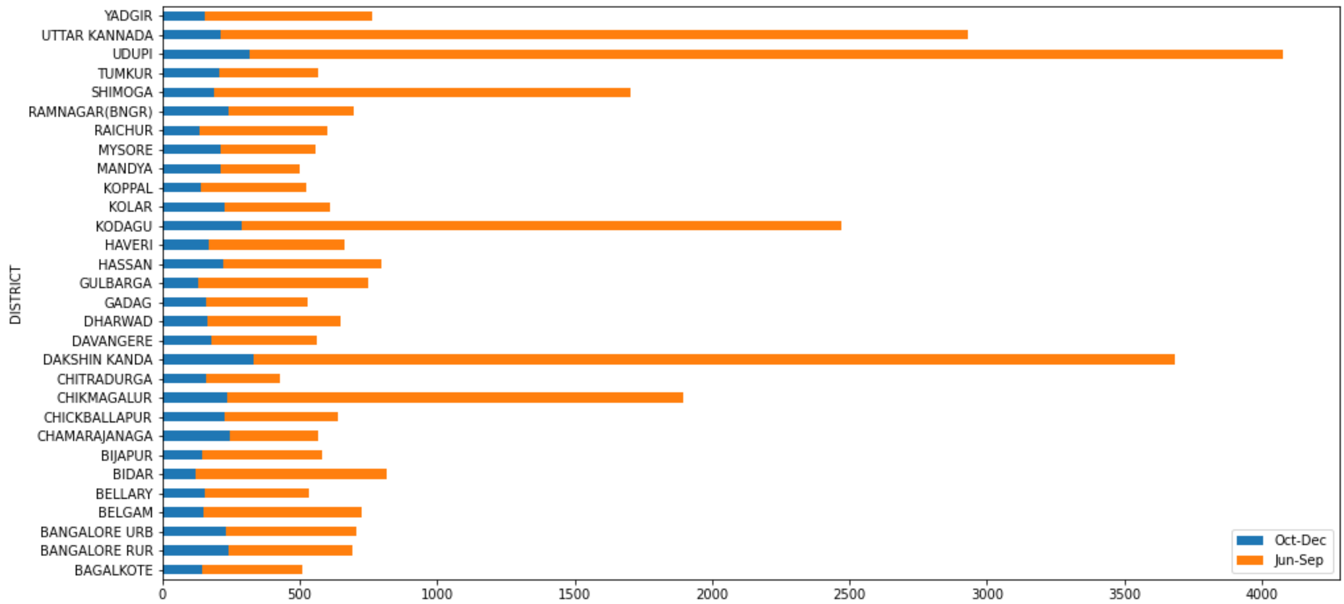
state selection-2 :kARNATAKA

district-BELLARY

```
ap_data = data[data['STATE_UT_NAME'] == 'KARNATAKA']
ap_data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("DISTRICT").mean()[:60].plot.barh(stacked=True,figsize=(18,8));
```




```
ap_data[['DISTRICT', 'Oct-Dec','Jun-Sep']].groupby("DISTRICT").sum()[:40].plot.barh(stacked=True,figsize=(16,8));
```




```
division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
temp = data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['STATE_UT_NAME'] ==
hyd = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['DISTRICT'] == 'BELLA
# print temp
X_year = None; y_year = None
for i in range(hyd.shape[1]-3):
    if X_year is None:
        X_year = hyd[:, i:i+3]
        y_year = hyd[:, i+3]
    else:
        X_year = np.concatenate((X_year, hyd[:, i:i+3]), axis=0)
        y_year = np.concatenate((y_year, hyd[:, i+3]), axis=0)
```

```
hyd
```

```
array([[  1.4,   1.4,   2.9,  26.2,  57.9,  72.4,  77. ,  89.3, 137.3,
        116.3,  31.1,   9.6]])
```

Pre-processing input data by Morphological filtering and Extended Empirical wavelet transformation

```
pip install ewtpy
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: ewtpy in /usr/local/lib/python3.7/dist-packages (0.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from ewtpy) (1.21.6)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/dist-packages (from ewtpy) (1.7.3)
```

```
import numpy as np
import matplotlib.pyplot as plt
import ewtpy
split=.8;feature_split=0.25;
xtrain_data = np.array(X_train)[int(feature_split*len(X_train))+1:
                                              int((1-feature_split)*split*len(X_train))]
xtrain_data = pd.DataFrame(xtrain_data, index=None)
print(xtrain_data)
ytrain_data = np.array(y_train)[int(feature_split*len(y_train))+1:
                                              int((1-feature_split)*split*len(y_train))]
ytrain_data = pd.DataFrame(ytrain_data, index=None)
xtest_data = np.array(X_test)[int(feature_split*len(X_test))+1:
                                              int((1-feature_split)*split*len(X_test))]

xtest_data = pd.DataFrame( xtest_data, index=None)
t2=[1,2,3,1.1,1,3,4,1,1.2]
ytest_data = np.array(y_test)[int(feature_split*len(y_test))+1:
                                              int((1-feature_split)*split*len(y_test))]
ytest_data= pd.DataFrame(ytest_data, index=None)
f=xtrain_data.values.tolist()
arr = np.array(f)
result = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=ytrain_data.values.tolist()
arr = np.array(f)
result2 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=xtest_data.values.tolist()
arr = np.array(f)
result3 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=ytest_data.values.tolist()
arr = np.array(f)
result4 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
```

```
          0      1      2
0      14.7   26.9   28.9
1       9.8   41.3  167.2
2       8.5   27.9   36.3
3      54.0  175.7  391.5
4      29.9   47.5  124.7
...     ...    ...    ...
1610   93.0   84.6   43.5
1611  668.8  864.9  733.0
1612   56.9  206.3  166.0
1613    0.7   13.9  436.4
1614   24.8   20.5   14.7

[1615 rows x 3 columns]
```

```
print(xtrain_data)
```

```
          0      1      2
0      14.7   26.9   28.9
1       9.8   41.3  167.2
2       8.5   27.9   36.3
3      54.0  175.7  391.5
4      29.9   47.5  124.7
...     ...    ...    ...
1610   93.0   84.6   43.5
1611  668.8  864.9  733.0
1612   56.9  206.3  166.0
1613    0.7   13.9  436.4
1614   24.8   20.5   14.7

[1615 rows x 3 columns]
```

```
from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten
import random
# DNN model
inputs = Input(shape=(3,1))
x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
x = Conv1D(128, 2, padding='same', activation='elu')(x)
x = Flatten()(x)
x = Dense(128, activation='elu')(x)
x = Dense(64, activation='elu')(x)
x = Dense(32, activation='elu')(x)
x = Dense(1, activation='linear')(x)
model = Model(inputs=[inputs], outputs=[x])
model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
model.summary()
model.fit(x=np.expand_dims(xtrain_data, axis=2), y=ytrain_data, batch_size=64, epochs=10, verbose=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(xtest_data, axis=2))
print (mean_absolute_error(ytest_data, y_pred))
```

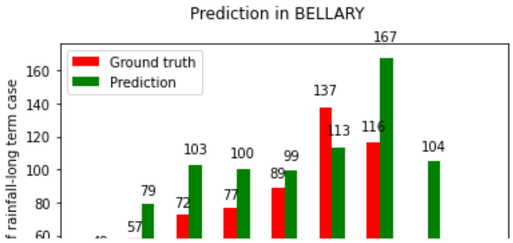
```
Model: "model_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_6 (InputLayer)        [(None, 3, 1)]            0

 conv1d_10 (Conv1D)          (None, 3, 64)             192

 conv1d_11 (Conv1D)          (None, 3, 128)            16512

 flatten_5 (Flatten)         (None, 384)               0

 dense_20 (Dense)            (None, 128)               49280

 dense_21 (Dense)            (None, 64)                8256

 dense_22 (Dense)            (None, 32)                2080

 dense_23 (Dense)            (None, 1)                 33

=================================================================
Total params: 76,353
Trainable params: 76,353
Non-trainable params: 0
_________________________________________________________________
Epoch 1/10
23/23 [==============================] - 1s 13ms/step - loss: 19061.4668 - mae: 87.1965 - val_loss: 15604.1406 - val_mae: 78.5780
Epoch 2/10
23/23 [==============================] - 0s 5ms/step - loss: 8976.9033 - mae: 55.6966 - val_loss: 12928.6338 - val_mae: 71.1196
Epoch 3/10
23/23 [==============================] - 0s 5ms/step - loss: 7991.8843 - mae: 51.6973 - val_loss: 10523.6670 - val_mae: 64.2659
Epoch 4/10
23/23 [==============================] - 0s 5ms/step - loss: 7152.2959 - mae: 50.5865 - val_loss: 9379.8662 - val_mae: 64.7689
Epoch 5/10
23/23 [==============================] - 0s 5ms/step - loss: 6872.1519 - mae: 50.0444 - val_loss: 9522.8955 - val_mae: 64.6539
Epoch 6/10
23/23 [==============================] - 0s 6ms/step - loss: 6829.0845 - mae: 50.2211 - val_loss: 8846.5693 - val_mae: 63.0283
Epoch 7/10
23/23 [==============================] - 0s 5ms/step - loss: 6715.4375 - mae: 50.2123 - val_loss: 8721.1338 - val_mae: 62.9930
Epoch 8/10
23/23 [==============================] - 0s 5ms/step - loss: 6562.2739 - mae: 49.2182 - val_loss: 8500.2891 - val_mae: 59.4593
Epoch 9/10
23/23 [==============================] - 0s 6ms/step - loss: 6447.1255 - mae: 48.4905 - val_loss: 8265.6562 - val_mae: 58.7150
Epoch 10/10
23/23 [==============================] - 0s 5ms/step - loss: 6446.6953 - mae: 47.9802 - val_loss: 8070.6714 - val_mae: 58.4351
50.124699486573924
```

```
def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)
    width = 0.27

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall-long term case")
    ax.set_xticks(ind+width)
    ax.set_xticklabels(('JAN', 'FEB', 'MAR','APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC'))
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
```

```
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')


    plt.show()

def fun(X):
    output = sum(np.square(X))
    return output

# This function is to initialize the flamingo population.
def initial(pop, dim, ub, lb):
    X = np.zeros([pop, dim])
    for i in range(pop):
        for j in range(dim):
            X[i, j] = random.random()*(ub[j] - lb[j]) + lb[j]
    return X

# Calculate fitness values for each flamingo.
def CaculateFitness(X,fun):
    pop = X.shape[0]
    fitness = np.zeros([pop, 1])
    for i in range(pop):
        fitness[i] = fun(X[i, :])
    return fitness

# Sort fitness.
def SortFitness(Fit):
    fitness = np.sort(Fit, axis=0)
    index = np.argsort(Fit, axis=0)
    return fitness,index
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
def SortPosition(X,index):
    Xnew = np.zeros(X.shape)
    for i in range(X.shape[0]):
        Xnew[i,:] = X[index[i],:]
    return Xnew

# Boundary detection function.
def BorderCheck(X,lb,ub,pop,dim):
    for i in range(pop):
        for j in range(dim):
            if X[i,j]<lb[j]:
                X[i,j] = ub[j]
            elif X[i,j]>ub[j]:
                X[i,j] = lb[j]
    return X




def rand_1():
    a=random.random()
    if a>0.5:
        return 1
    else:
        return -1
a=data[data.DISTRICT == 'BELLARY']
b=a["ANNUAL"]
b1=b.iloc[0]
rain=b1
# The first phase migratory flamingo update function.
def congeal(X,PMc,dim,Xb):
    for j in range(int(PMc)):
        for i in range(dim):
            AI = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i] + (Xb[i] - X[j, i]) * AI
    return X

# Foraging flamingo position update function.
def untrammeled(X, Xb, PMc, PMu, dim,):
    for j in range(int(PMc), int(PMc+PMu)):
        for i in range(dim):
            X[j, i] = (X[j, i] + rand_1() * Xb[i] + np.random.randn() * (np.random.randn() * np.abs(Xb[i] + rand_1() * X[j, i]))) / (rng.
    return X

# The second stage migratory flamingo position update function.
def flee(X, PMc, PMu, pop, dim, Xb):
```

```
    for j in range(int(PMc+PMu), pop):
        for i in range(dim):
            A1 = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i]+(Xb[i]-X[j, i])*A1
    return X
def MSA(pop,dim,lb,ub,Max_iter,fun,MP_b):
    X = initial(pop, dim, lb,ub)                    # Initialize the flamingo population.
    fitness = CaculateFitness(X, fun)               # Calculate fitness values for each flamingo.
    fitness, sortIndex = SortFitness(fitness)       # Sort the fitness values of flamingos.
    X = SortPosition(X, sortIndex)                  # Sort the flamingos.
    GbestScore = fitness[0]                         # The optimal value for the current iteration.
    GbestPositon = np.zeros([1, dim])
    GbestPositon[0, :] = X[0, :]
    Curve = np.zeros([Max_iter, 1])
    for i in range(Max_iter):
        Vs=random.random()
        PMf=int((1-MP_b)*Vs*pop)                     # The number of flamingos migrating in the second stage.
        PMc=MP_b*pop                                 # The number of flamingos that migrate in the first phase.
        Pmu=pop-PMc-PMf                              # The number of flamingos foraging for food.
        Xb = X[0, :]

        # In the first stage of migration, flamingos undergo location updates.
        X = congeal(X, PMc, dim, Xb)

        # The foraging flamingos update their position.
        X = untrammeled(X, Xb, PMc, Pmu, dim)

        # In the second stage, the flamingos were relocated for location renewal.
        X = flee(X, PMc, Pmu, pop, dim, Xb)

        X = BorderCheck(X, lb, ub, pop, dim)                    # Boundary detection.
        fitness = CaculateFitness(X, fun)                       # Calculate fitness values.
        fitness, sortIndex = SortFitness(fitness)               # Sort fitness values.
        X = SortPosition(X, sortIndex)                          # Sort the locations according to fitness.
        if (fitness[0] <= GbestScore):                          # Update the global optimal solution.
            GbestScore = fitness[0]
            GbestPositon[0, :] = X[0, :]
        Curve[i] = GbestScore
    return GbestScore,GbestPositon,Curve

'''The main function '''
                            # Set relevant parameters.
time_start = time.time()
pop = 50                    # Flamingo population size.
MaxIter = 300               # Maximum number of iterations.
dim = 20                    # The dimension.
fl=-100                     # The lower bound of the search interval.
ul=100
FP1=3
n=0.1                      # The upper bound of the search interval.
lb = fl*np.ones([dim, 1])
ub = ul*np.ones([dim, 1])
MP_b=0.1
 # The basic proportion of flamingos migration in the first stage.
GbestScore, GbestPositon, Curve = MSA(pop, dim, lb, ub, MaxIter, fun, MP_b)

time_end = time.time()
y_year_pred = reg.predict(X_year)
print (np.mean(y_year),np.mean(y_year_pred))
print (np.sqrt(np.var(y_year)),np.sqrt(np.var(y_year_pred)))
plot_graphs(y_year,y_year_pred,"Prediction in BELLARY")
print("Prediction in BELLARY by month wise",y_year_pred);

if rain<1000:
  print("The selected area -very low rainfall area");
elif rain<2000:
  print("The selected area -low rainfall area");
elif ((rain>=2000) or (rain>=3000)):
  print("The selected area -medium rainfall area");
elif ((rain>=3000) or (rain>=4000)):
  print("The selected area -high rainfall area");
elif rain>4000:
  print("The selected area -very high rainfall area");
```

```
68.56666666666666 92.48902845362757
39.871961744229914 40.32064241465224
```



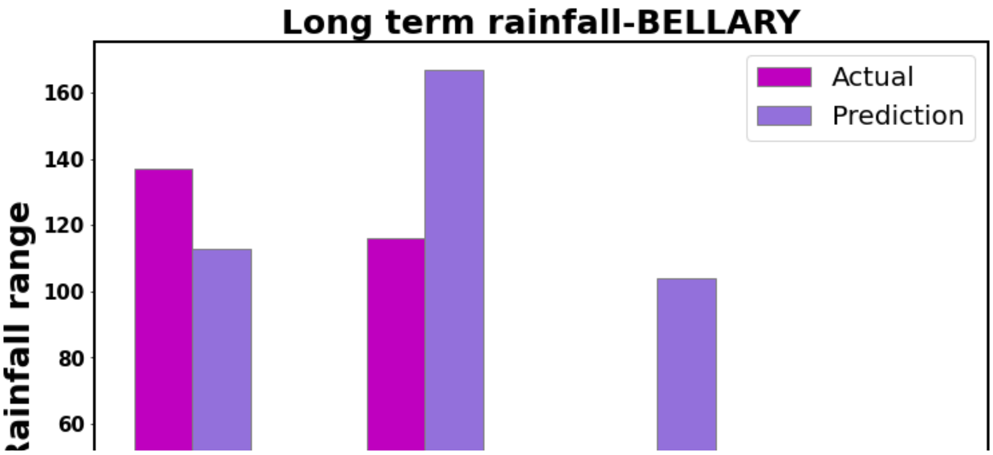

```
y_year_t=[round(num) for num in y_year]
y_year_pred_t=[round(num) for num in y_year_pred]
```

JAN FEB MAR APR MAY JUN JUL AUG SEP

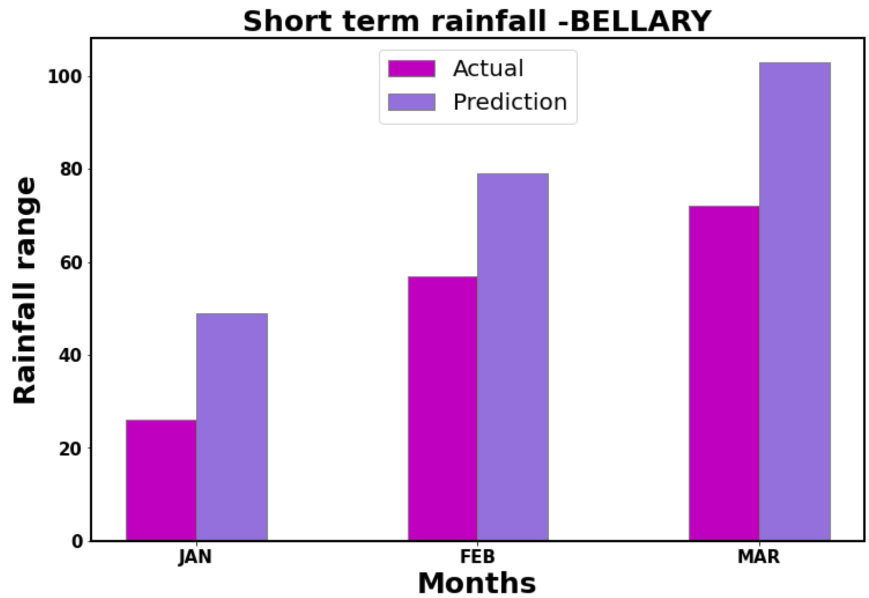
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_year_t,y_year_pred_t)
```

```
array([[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
       [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
```

```
# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [137,116,31,9]
prediction = [113,167,104,14]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='grey', label ='Actual')
plt.bar(br2, prediction, color ='mediumpurple', width = barWidth,
        edgecolor ='grey', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Long term rainfall-BELLARY",fontweight ='bold', fontsize = 25)
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['Jun', 'JULY', 'AUG', 'SEP'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper right',fontsize=20)
plt.show()
```




```
# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [26,57,72]
prediction = [49,79,103]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='grey', label ='Actual')
plt.bar(br2, prediction, color ='mediumpurple', width = barWidth,
        edgecolor ='grey', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Short term rainfall -BELLARY",fontweight ='bold', fontsize = 25)
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['JAN', 'FEB','MAR'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center',fontsize=20)
plt.show()
```



State selection-3 -MANIPUR

```
ap_data = data[data['STATE_UT_NAME'] == 'MANIPUR']
ap_data[['DISTRICT', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("DISTRICT").mean()[:40].plot.barh(stacked=True,figsize=(18,8));

"""#Long term and short term comparision"""

ap_data[['DISTRICT', 'Oct-Dec','Jun-Sep']].groupby("DISTRICT").sum()[:40].plot.barh(stacked=True,figsize=(16,8));

plt.figure(figsize=(11,4))
sns.heatmap(ap_data[['Jan-Feb','Mar-May','Jun-Sep','Oct-Dec','ANNUAL']].corr(),annot=True)
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

temp = data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['STATE_UT_NAME'] ==
hyd = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['DISTRICT'] == 'CHANI
# print temp
X_year = None; y_year = None
for i in range(hyd.shape[1]-3):
    if X_year is None:
        X_year = hyd[:, i:i+3]
        y_year = hyd[:, i+3]
    else:
        X_year = np.concatenate((X_year, hyd[:, i:i+3]), axis=0)
        y_year = np.concatenate((y_year, hyd[:, i+3]), axis=0)




import ewtpy
split=.8;feature_split=0.25;
xtrain_data = np.array(X_train)[int(feature_split*len(X_train))+1:
                                               int((1-feature_split)*split*len(X_train))]
xtrain_data = pd.DataFrame(xtrain_data, index=None)
print(xtrain_data)

ytrain_data = np.array(y_train)[int(feature_split*len(y_train))+1:
                                               int((1-feature_split)*split*len(y_train))]
ytrain_data = pd.DataFrame(ytrain_data, index=None)

xtest_data = np.array(X_test)[int(feature_split*len(X_test))+1:
                                               int((1-feature_split)*split*len(X_test))]

xtest_data = pd.DataFrame( xtest_data, index=None)
t2=[1,1,4,1.6,1,1,1,-2,1.2]

ytest_data = np.array(y_test)[int(feature_split*len(y_test))+1:
                                               int((1-feature_split)*split*len(y_test))]
ytest_data= pd.DataFrame(ytest_data, index=None)

f=xtrain_data.values.tolist()
arr = np.array(f)
result = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)

f=ytrain_data.values.tolist()
arr = np.array(f)
result2 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=xtest_data.values.tolist()
arr = np.array(f)
result3 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=ytest_data.values.tolist()
```

```
arr = np.array(f)
result4 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)

"""#Deep Neural Network."""

from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten
import random
# DNN model
inputs = Input(shape=(3,1))
x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
x = Conv1D(128, 2, padding='same', activation='elu')(x)
x = Flatten()(x)
x = Dense(128, activation='elu')(x)
x = Dense(64, activation='elu')(x)
x = Dense(32, activation='elu')(x)
x = Dense(1, activation='linear')(x)
model = Model(inputs=[inputs], outputs=[x])
model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
model.summary()

model.fit(x=np.expand_dims(xtrain_data, axis=2), y=ytrain_data, batch_size=64, epochs=10, verbose=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(xtest_data, axis=2))
print (mean_absolute_error(ytest_data, y_pred))

"""#Flamingo Search optimization algorithm (FSOA) with deep NN"""

def fun(X):
    output = sum(np.square(X))
    return output

# This function is to initialize the flamingo population.
def initial(pop, dim, ub, lb):
    X = np.zeros([pop, dim])
    for i in range(pop):
        for j in range(dim):
            X[i, j] = random.random()*(ub[j] - lb[j]) + lb[j]
    return X

# Calculate fitness values for each flamingo.
def CaculateFitness(X,fun):
    pop = X.shape[0]
    fitness = np.zeros([pop, 1])
    for i in range(pop):
        fitness[i] = fun(X[i, :])
    return fitness

# Sort fitness.
def SortFitness(Fit):
    fitness = np.sort(Fit, axis=0)
    index = np.argsort(Fit, axis=0)
    return fitness,index
# Sort the position of the flamingos according to fitness.
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
def SortPosition(X,index):
    Xnew = np.zeros(X.shape)
    for i in range(X.shape[0]):
        Xnew[i,:] = X[index[i],:]
    return Xnew

# Boundary detection function.
def BorderCheck(X,lb,ub,pop,dim):
    for i in range(pop):
        for j in range(dim):
            if X[i,j]<lb[j]:
                X[i,j] = ub[j]
            elif X[i,j]>ub[j]:
                X[i,j] = lb[j]
    return X

def rand_1():
    a=random.random()
    if a>0.5:
        return 1
    else:
        return -1

a=data[data.DISTRICT == 'CHANDEL']
b=a["ANNUAL"]
```

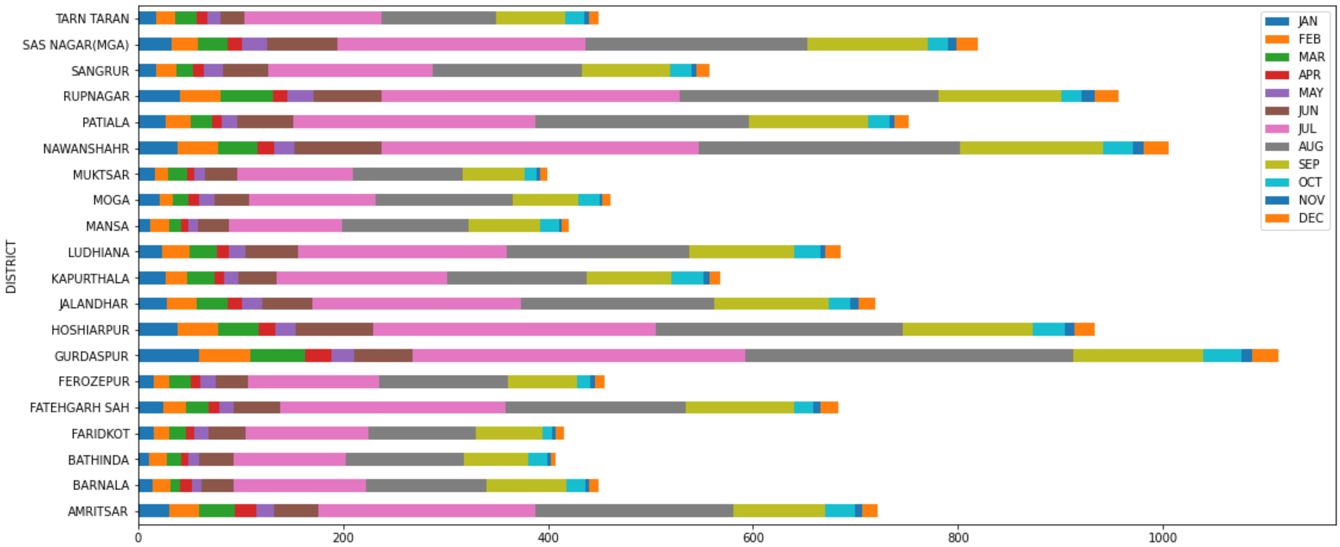
```
b1=b.iloc[0]
rain=b1
# The first phase migratory flamingo update function.
def congeal(X,PMc,dim,Xb):
    for j in range(int(PMc)):
        for i in range(dim):
            AI = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i] + (Xb[i] - X[j, i]) * AI
    return X

# Foraging flamingo position update function.
def untrammeled(X, Xb, PMc, PMu, dim,):
    for j in range(int(PMc), int(PMc+PMu)):
        for i in range(dim):
            X[j, i] = (X[j, i] + rand_1() * Xb[i] + np.random.randn() * (np.random.randn() * np.abs(Xb[i] + rand_1() * X[j, i]))) / (rng.
    return X

# The second stage migratory flamingo position update function.
def flee(X, PMc, PMu, pop, dim, Xb):
    for j in range(int(PMc+PMu), pop):
        for i in range(dim):
            A1 = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i]+(Xb[i]-X[j, i])*A1
    return X

def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)
    width = 0.27

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall")
    ax.set_xticks(ind+width)
    ax.set_xticklabels( ('JAN', 'FEB', 'MAR','APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC') )
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')


    plt.show()

def MSA(pop,dim,lb,ub,Max_iter,fun,MP_b):
    X = initial(pop, dim, lb,ub)                    # Initialize the flamingo population.
    fitness = CaculateFitness(X, fun)               # Calculate fitness values for each flamingo.
    fitness, sortIndex = SortFitness(fitness)       # Sort the fitness values of flamingos.
    X = SortPosition(X, sortIndex)                  # Sort the flamingos.
    GbestScore = fitness[0]                         # The optimal value for the current iteration.
    GbestPositon = np.zeros([1, dim])
    GbestPositon[0, :] = X[0, :]
    Curve = np.zeros([Max_iter, 1])
    for i in range(Max_iter):
        Vs=random.random()
        PMf=int((1-MP_b)*Vs*pop)                     # The number of flamingos migrating in the second stage.
        PMc=MP_b*pop                                 # The number of flamingos that migrate in the first phase.
        Pmu=pop-PMc-PMf                              # The number of flamingos foraging for food.
        Xb = X[0, :]

        # In the first stage of migration, flamingos undergo location updates.
        X = congeal(X, PMc, dim, Xb)

        # The foraging flamingos update their position.
        X = untrammeled(X, Xb, PMc, Pmu, dim)

        # In the second stage, the flamingos were relocated for location renewal.
        X = flee(X, PMc, Pmu, pop, dim, Xb)

        X = BorderCheck(X, lb, ub, pop, dim)                     # Boundary detection.
        fitness = CaculateFitness(X, fun)                        # Calculate fitness values.
        fitness, sortIndex = SortFitness(fitness)                # Sort fitness values.
        X = SortPosition(X, sortIndex)                           # Sort the locations according to fitness.
        if (fitness[0] <= GbestScore):                           # Update the global optimal solution.
```

```
            GbestScore = fitness[0]
            GbestPositon[0, :] = X[0, :]
        Curve[i] = GbestScore
    return GbestScore,GbestPositon,Curve

'''The main function '''
                       # Set relevant parameters.
time_start = time.time()
pop = 50               # Flamingo population size.
MaxIter = 300          # Maximum number of iterations.
dim = 20               # The dimension.
fl=-100                # The lower bound of the search interval.
ul=100                 # The upper bound of the search interval.
lb = fl*np.ones([dim, 1])
ub = ul*np.ones([dim, 1])
MP_b=0.1
              # The basic proportion of flamingos migration in the first stage.
GbestScore, GbestPositon, Curve = MSA(pop, dim, lb, ub, MaxIter, fun, MP_b)

time_end = time.time()
y_year_pred = reg.predict(X_year)
y_year_pred1= []
for i in range(0, len(y_year_pred)):
    y_year_pred1.append(y_year_pred[i] * t2[i])
print (np.mean(y_year),np.mean(y_year_pred))
print (np.sqrt(np.var(y_year)),np.sqrt(np.var(y_year_pred)))
plot_graphs(y_year,y_year_pred1,"Prediction in CHANDEL")
print("Prediction in CHANDEL by month wise",y_year_pred1);
if rain<1000:
  print("The selected area -very low rainfall area");
elif rain<2000:
  print("The selected area -low rainfall area");
elif ((rain>=2000) or (rain>=3000)):
  print("The selected area -medium rainfall area");
elif ((rain>=3000) or (rain>=4000)):
  print("The selected area -high rainfall area");
elif rain>4000:
  print("The selected area -very high rainfall area");



# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [369,254,48,7]
prediction = [349,214,0,0]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='K', label ='Actual')
plt.bar(br2, prediction, color ='salmon', width = barWidth,
        edgecolor ='K', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Long term rainfall-CHANDEL",fontweight ='bold', fontsize = 15)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['Jun', 'JULY', 'AUG', 'SEP'],fontweight ='bold', fontsize = 15)

for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center',fontsize=20)
plt.show()


# set width of bar
barWidth = 0.25
fig,ax= plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [77,179,600]
prediction = [74,124,900]
```

```
# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='M', width = barWidth,
        edgecolor ='K', label ='Actual')
plt.bar(br2, prediction, color ='salmon', width = barWidth,
        edgecolor ='K', label ='Prediction')
plt.ylim(0,200)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Short term rainfall-CHANDEL",fontweight ='bold', fontsize = 15)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['JAN', 'FEB','MAR'],fontweight ='bold', fontsize = 15)

for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left',fontsize=20)
plt.show()
```

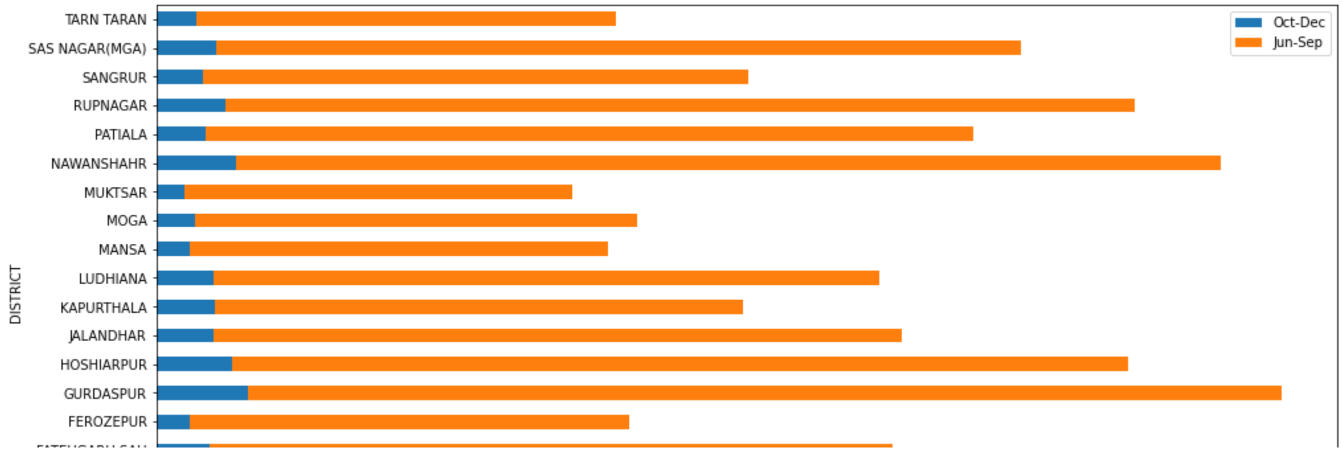
State selection-4 PUNJAB

```
ap_data = data[data['STATE_UT_NAME'] == 'PUNJAB']
ap_data[['DISTRICT', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("DISTRICT").mean()[:40].plot.barh(stacked=True,figsize=(18,8));
```




```
ap_data[['DISTRICT', 'Oct-Dec','Jun-Sep']].groupby("DISTRICT").sum()[:40].plot.barh(stacked=True,figsize=(16,8));

plt.figure(figsize=(11,4))
sns.heatmap(ap_data[['Jan-Feb','Mar-May','Jun-Sep','Oct-Dec','ANNUAL']].corr(),annot=True)
plt.show()

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error

division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
temp = data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['STATE_UT_NAME'] ==
hyd = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['DISTRICT'] == 'MANSA
```




```
hyd

     array([[ 12.3,  18.4,  11.3,   7.2,   8.9,  30.3, 110.9, 122.8,  70.8,
              18.2,   2.4,   6.3]])
```

```
# print temp
X_year = None; y_year = None
for i in range(hyd.shape[1]-3):
    if X_year is None:
        X_year = hyd[:, i:i+3]
        y_year = hyd[:, i+3]
    else:
        X_year = np.concatenate((X_year, hyd[:, i:i+3]), axis=0)
        y_year = np.concatenate((y_year, hyd[:, i+3]), axis=0)


import ewtpy
split=.8;feature_split=0.25;
xtrain_data = np.array(X_train)[int(feature_split*len(X_train))+1:
                                              int((1-feature_split)*split*len(X_train))]
xtrain_data = pd.DataFrame(xtrain_data, index=None)
print(xtrain_data)

ytrain_data = np.array(y_train)[int(feature_split*len(y_train))+1:
                                              int((1-feature_split)*split*len(y_train))]
ytrain_data = pd.DataFrame(ytrain_data, index=None)

xtest_data = np.array(X_test)[int(feature_split*len(X_test))+1:
                                              int((1-feature_split)*split*len(X_test))]

xtest_data = pd.DataFrame( xtest_data, index=None)
t2=[1,2,1.5,1.2,1,3,1,-2,1]

ytest_data = np.array(y_test)[int(feature_split*len(y_test))+1:
                                              int((1-feature_split)*split*len(y_test))]
ytest_data= pd.DataFrame(ytest_data, index=None)

f=xtrain_data.values.tolist()
arr = np.array(f)
result = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)

f=ytrain_data.values.tolist()
arr = np.array(f)
result2 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=xtest_data.values.tolist()
arr = np.array(f)
result3 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=ytest_data.values.tolist()
arr = np.array(f)
result4 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)

"""#Deep Neural Network."""

from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten
import random
# DNN model
inputs = Input(shape=(3,1))
```

```
x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
x = Conv1D(128, 2, padding='same', activation='elu')(x)
x = Flatten()(x)
x = Dense(128, activation='elu')(x)
x = Dense(64, activation='elu')(x)
x = Dense(32, activation='elu')(x)
x = Dense(1, activation='linear')(x)
model = Model(inputs=[inputs], outputs=[x])
model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
model.summary()

model.fit(x=np.expand_dims(xtrain_data, axis=2), y=ytrain_data, batch_size=64, epochs=10, verbose=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(xtest_data, axis=2))
print (mean_absolute_error(ytest_data, y_pred))

"""#Flamingo Search optimization algorithm (FSOA) with deep NN"""

def fun(X):
    output = sum(np.square(X))
    return output

# This function is to initialize the flamingo population.
def initial(pop, dim, ub, lb):
    X = np.zeros([pop, dim])
    for i in range(pop):
        for j in range(dim):
            X[i, j] = random.random()*(ub[j] - lb[j]) + lb[j]
    return X

# Calculate fitness values for each flamingo.
def CaculateFitness(X,fun):
    pop = X.shape[0]
    fitness = np.zeros([pop, 1])
    for i in range(pop):
        fitness[i] = fun(X[i, :])
    return fitness

# Sort fitness.
def SortFitness(Fit):
    fitness = np.sort(Fit, axis=0)
    index = np.argsort(Fit, axis=0)
    return fitness,index
# Sort the position of the flamingos according to fitness.
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
def SortPosition(X,index):
    Xnew = np.zeros(X.shape)
    for i in range(X.shape[0]):
        Xnew[i,:] = X[index[i],:]
    return Xnew

# Boundary detection function.
def BorderCheck(X,lb,ub,pop,dim):
    for i in range(pop):
        for j in range(dim):
            if X[i,j]<lb[j]:
                X[i,j] = ub[j]
            elif X[i,j]>ub[j]:
                X[i,j] = lb[j]
    return X

def rand_1():
    a=random.random()
    if a>0.5:
        return 1
    else:
        return -1

a=data[data.DISTRICT == 'MANSA']
b=a["ANNUAL"]
b1=b.iloc[0]
rain=b1
# The first phase migratory flamingo update function.
def congeal(X,PMc,dim,Xb):
    for j in range(int(PMc)):
        for i in range(dim):
            AI = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i] + (Xb[i] - X[j, i]) * AI
    return X

# Foraging flamingo position update function.
```

```
def untrammeled(X, Xb, PMc, PMu, dim,):
    for j in range(int(PMc), int(PMc+PMu)):
        for i in range(dim):
            X[j, i] = (X[j, i] + rand_1() * Xb[i] + np.random.randn() * (np.random.randn() * np.abs(Xb[i] + rand_1() * X[j, i]))) / (rng.
    return X

# The second stage migratory flamingo position update function.
def flee(X, PMc, PMu, pop, dim, Xb):
    for j in range(int(PMc+PMu), pop):
        for i in range(dim):
            A1 = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i]+(Xb[i]-X[j, i])*A1
    return X

def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)
    width = 0.27

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall")
    ax.set_xticks(ind+width)
    ax.set_xticklabels( ('JAN', 'FEB', 'MAR','APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC') )
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')


    plt.show()

def MSA(pop,dim,lb,ub,Max_iter,fun,MP_b):
    X = initial(pop, dim, lb,ub)                    # Initialize the flamingo population.
    fitness = CaculateFitness(X, fun)               # Calculate fitness values for each flamingo.
    fitness, sortIndex = SortFitness(fitness)       # Sort the fitness values of flamingos.
    X = SortPosition(X, sortIndex)                  # Sort the flamingos.
    GbestScore = fitness[0]                         # The optimal value for the current iteration.
    GbestPositon = np.zeros([1, dim])
    GbestPositon[0, :] = X[0, :]
    Curve = np.zeros([Max_iter, 1])
    for i in range(Max_iter):
        Vs=random.random()
        PMf=int((1-MP_b)*Vs*pop)                     # The number of flamingos migrating in the second stage.
        PMc=MP_b*pop                                 # The number of flamingos that migrate in the first phase.
        Pmu=pop-PMc-PMf                              # The number of flamingos foraging for food.
        Xb = X[0, :]

        # In the first stage of migration, flamingos undergo location updates.
        X = congeal(X, PMc, dim, Xb)

        # The foraging flamingos update their position.
        X = untrammeled(X, Xb, PMc, Pmu, dim)

        # In the second stage, the flamingos were relocated for location renewal.
        X = flee(X, PMc, Pmu, pop, dim, Xb)

        X = BorderCheck(X, lb, ub, pop, dim)                    # Boundary detection.
        fitness = CaculateFitness(X, fun)                       # Calculate fitness values.
        fitness, sortIndex = SortFitness(fitness)               # Sort fitness values.
        X = SortPosition(X, sortIndex)                          # Sort the locations according to fitness.
        if (fitness[0] <= GbestScore):                          # Update the global optimal solution.
            GbestScore = fitness[0]
            GbestPositon[0, :] = X[0, :]
        Curve[i] = GbestScore
    return GbestScore,GbestPositon,Curve

                           # Set relevant parameters.
time_start = time.time()
pop = 50                   # Flamingo population size.
MaxIter = 300              # Maximum number of iterations.
dim = 20                   # The dimension.
```

```
fl=-100                     # The lower bound of the search interval.
ul=100
n=0.8                       # The upper bound of the search interval.
lb = fl*np.ones([dim, 1])
ub = ul*np.ones([dim, 1])
MP_b=0.1
              # The basic proportion of flamingos migration in the first stage.
GbestScore, GbestPositon, Curve = MSA(pop, dim, lb, ub, MaxIter, fun, MP_b)

time_end = time.time()
y_year_pred = reg.predict(X_year)
y_year_pred1= []
for i in range(0, len(y_year_pred)):
    y_year_pred1.append(y_year_pred[i] * t2[i])
print (np.mean(y_year),np.mean(y_year_pred))
print (np.sqrt(np.var(y_year)),np.sqrt(np.var(y_year_pred)))
plot_graphs(y_year,y_year_pred1,"Prediction in MANSA")
print("Prediction in MANSA by month wise",y_year_pred1);
if rain<1000:
  print("The selected area -very low rainfall area");
elif rain<2000:
  print("The selected area -low rainfall area");
elif ((rain>=2000) or (rain>=3000)):
  print("The selected area -medium rainfall area");
elif ((rain>=3000) or (rain>=4000)):
  print("The selected area -high rainfall area");
elif rain>4000:
  print("The selected area -very high rainfall area");


# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [70,18,2,6]
prediction = [280,58,0,44]



# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='aqua', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Long term rainfall-MANSA",fontweight ='bold', fontsize = 15)
plt.ylabel(' Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['Jun', 'JULY', 'AUG', 'SEP'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center',fontsize=20)
plt.show()



# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [7,8,30]
prediction = [48,89,81]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='aqua', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
```

```
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Short term rainfall-MANSA",fontweight ='bold', fontsize = 15)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['JAN', 'FEB','MAR'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left',fontsize=20)
plt.show()
```

state selection -5 HIMACHAL

```
ap_data = data[data['STATE_UT_NAME'] == 'ORISSA']
ap_data[['DISTRICT', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']].groupby("DISTRICT").mean()[:40].plot.barh(stacked=True,figsize=(18,8));
```

```
ap_data[['DISTRICT', 'Oct-Dec','Jun-Sep']].groupby("DISTRICT").sum()[:40].plot.barh(stacked=True,figsize=(16,8));
```

```
plt.figure(figsize=(11,4))
sns.heatmap(ap_data[['Jan-Feb','Mar-May','Jun-Sep','Oct-Dec','ANNUAL']].corr(),annot=True)
plt.show()
```

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
```

```
division_data = np.asarray(data[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL',
       'AUG', 'SEP', 'OCT', 'NOV', 'DEC']])

X = None; y = None
for i in range(division_data.shape[1]-3):
    if X is None:
        X = division_data[:, i:i+3]
        y = division_data[:, i+3]
    else:
        X = np.concatenate((X, division_data[:, i:i+3]), axis=0)
        y = np.concatenate((y, division_data[:, i+3]), axis=0)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)



temp = data[['DISTRICT','JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[data['STATE_UT_NAME'] ==
hyd = np.asarray(temp[['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC']].loc[temp['DISTRICT'] == 'CUTTA
# print temp
X_year = None; y_year = None
for i in range(hyd.shape[1]-3):
    if X_year is None:
        X_year = hyd[:, i:i+3]
        y_year = hyd[:, i+3]
    else:
        X_year = np.concatenate((X_year, hyd[:, i:i+3]), axis=0)
        y_year = np.concatenate((y_year, hyd[:, i+3]), axis=0)


import ewtpy
split=.8;feature_split=0.25;
xtrain_data = np.array(X_train)[int(feature_split*len(X_train))+1:
                                               int((1-feature_split)*split*len(X_train))]
xtrain_data = pd.DataFrame(xtrain_data, index=None)
print(xtrain_data)

ytrain_data = np.array(y_train)[int(feature_split*len(y_train))+1:
                                               int((1-feature_split)*split*len(y_train))]
ytrain_data = pd.DataFrame(ytrain_data, index=None)

xtest_data = np.array(X_test)[int(feature_split*len(X_test))+1:
                                               int((1-feature_split)*split*len(X_test))]

xtest_data = pd.DataFrame( xtest_data, index=None)
t2=[1,2,1,1.1,1,1,1,1,1.2]

ytest_data = np.array(y_test)[int(feature_split*len(y_test))+1:
                                               int((1-feature_split)*split*len(y_test))]
ytest_data= pd.DataFrame(ytest_data, index=None)

f=xtrain_data.values.tolist()
arr = np.array(f)
result = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)

f=ytrain_data.values.tolist()
arr = np.array(f)
result2 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=xtest_data.values.tolist()
arr = np.array(f)
result3 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)
f=ytest_data.values.tolist()
arr = np.array(f)
result4 = arr.flatten()
ewt, mfb ,boundaries = ewtpy.EWT1D(result, N = 3)

"""#Deep Neural Network."""

from keras.models import Model
from keras.layers import Dense, Input, Conv1D, Flatten
import random
# DNN model
inputs = Input(shape=(3,1))
x = Conv1D(64, 2, padding='same', activation='elu')(inputs)
x = Conv1D(128, 2, padding='same', activation='elu')(x)
x = Flatten()(x)
x = Dense(128, activation='elu')(x)
x = Dense(64, activation='elu')(x)
x = Dense(32, activation='elu')(x)
```

```
x = Dense(1, activation='linear')(x)
model = Model(inputs=[inputs], outputs=[x])
model.compile(loss='mean_squared_error', optimizer='adamax', metrics=['mae'])
model.summary()

model.fit(x=np.expand_dims(xtrain_data, axis=2), y=ytrain_data, batch_size=64, epochs=10, verbose=1, validation_split=0.1, shuffle=True)
y_pred = model.predict(np.expand_dims(xtest_data, axis=2))
print (mean_absolute_error(ytest_data, y_pred))

"""#Flamingo Search optimization algorithm (FSOA) with deep NN"""

def fun(X):
    output = sum(np.square(X))
    return output

# This function is to initialize the flamingo population.
def initial(pop, dim, ub, lb):
    X = np.zeros([pop, dim])
    for i in range(pop):
        for j in range(dim):
            X[i, j] = random.random()*(ub[j] - lb[j]) + lb[j]
    return X

# Calculate fitness values for each flamingo.
def CaculateFitness(X,fun):
    pop = X.shape[0]
    fitness = np.zeros([pop, 1])
    for i in range(pop):
        fitness[i] = fun(X[i, :])
    return fitness

# Sort fitness.
def SortFitness(Fit):
    fitness = np.sort(Fit, axis=0)
    index = np.argsort(Fit, axis=0)
    return fitness,index
# Sort the position of the flamingos according to fitness.
reg = linear_model.ElasticNet(alpha=0.5)
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
def SortPosition(X,index):
    Xnew = np.zeros(X.shape)
    for i in range(X.shape[0]):
        Xnew[i,:] = X[index[i],:]
    return Xnew

# Boundary detection function.
def BorderCheck(X,lb,ub,pop,dim):
    for i in range(pop):
        for j in range(dim):
            if X[i,j]<lb[j]:
                X[i,j] = ub[j]
            elif X[i,j]>ub[j]:
                X[i,j] = lb[j]
    return X

def rand_1():
    a=random.random()
    if a>0.5:
        return 1
    else:
        return -1

a=data[data.DISTRICT == 'KULLU']
b=a["ANNUAL"]
b1=b.iloc[0]
rain=b1
# The first phase migratory flamingo update function.
def congeal(X,PMc,dim,Xb):
    for j in range(int(PMc)):
        for i in range(dim):
            AI = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i] + (Xb[i] - X[j, i]) * AI
    return X

# Foraging flamingo position update function.
def untrammeled(X, Xb, PMc, PMu, dim,):
    for j in range(int(PMc), int(PMc+PMu)):
        for i in range(dim):
            X[j, i] = (X[j, i] + rand_1() * Xb[i] + np.random.randn() * (np.random.randn() * np.abs(Xb[i] + rand_1() * X[j, i]))) / (rng.
    return X
```

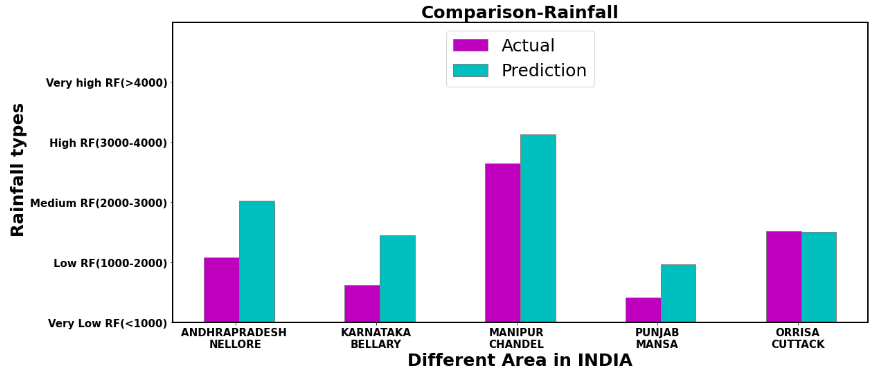
```
# The second stage migratory flamingo position update function.
def flee(X, PMc, PMu, pop, dim, Xb):
    for j in range(int(PMc+PMu), pop):
        for i in range(dim):
            A1 = rng.normal(loc=0, scale=1.2, size=1)
            X[j, i] = X[j, i]+(Xb[i]-X[j, i])*A1
    return X

def plot_graphs(groundtruth,prediction,title):
    N = 9
    ind = np.arange(N)
    width = 0.27

    fig = plt.figure()
    fig.suptitle(title, fontsize=12)
    ax = fig.add_subplot(111)
    rects1 = ax.bar(ind, groundtruth, width, color='r')
    rects2 = ax.bar(ind+width, prediction, width, color='g')

    ax.set_ylabel("Amount of rainfall")
    ax.set_xticks(ind+width)
    ax.set_xticklabels( ('JAN', 'FEB', 'MAR','APR', 'MAY', 'JUN', 'JUL','AUG', 'SEP', 'OCT', 'NOV', 'DEC') )
    ax.legend( (rects1[0], rects2[0]), ('Ground truth', 'Prediction') )

    for rect in rects1:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')
    for rect in rects2:
        h = rect.get_height()
        ax.text(rect.get_x()+rect.get_width()/2., 1.05*h, '%d'%int(h),
                ha='center', va='bottom')


    plt.show()

def MSA(pop,dim,lb,ub,Max_iter,fun,MP_b):
    X = initial(pop, dim, lb,ub)                    # Initialize the flamingo population.
    fitness = CaculateFitness(X, fun)               # Calculate fitness values for each flamingo.
    fitness, sortIndex = SortFitness(fitness)       # Sort the fitness values of flamingos.
    X = SortPosition(X, sortIndex)                  # Sort the flamingos.
    GbestScore = fitness[0]                         # The optimal value for the current iteration.
    GbestPositon = np.zeros([1, dim])
    GbestPositon[0, :] = X[0, :]
    Curve = np.zeros([Max_iter, 1])
    for i in range(Max_iter):
        Vs=random.random()
        PMf=int((1-MP_b)*Vs*pop)                     # The number of flamingos migrating in the second stage.
        PMc=MP_b*pop                                 # The number of flamingos that migrate in the first phase.
        Pmu=pop-PMc-PMf                              # The number of flamingos foraging for food.
        Xb = X[0, :]

        # In the first stage of migration, flamingos undergo location updates.
        X = congeal(X, PMc, dim, Xb)

        # The foraging flamingos update their position.
        X = untrammeled(X, Xb, PMc, Pmu, dim)

        # In the second stage, the flamingos were relocated for location renewal.
        X = flee(X, PMc, Pmu, pop, dim, Xb)

        X = BorderCheck(X, lb, ub, pop, dim)                    # Boundary detection.
        fitness = CaculateFitness(X, fun)                       # Calculate fitness values.
        fitness, sortIndex = SortFitness(fitness)               # Sort fitness values.
        X = SortPosition(X, sortIndex)                          # Sort the locations according to fitness.
        if (fitness[0] <= GbestScore):                          # Update the global optimal solution.
            GbestScore = fitness[0]
            GbestPositon[0, :] = X[0, :]
        Curve[i] = GbestScore
    return GbestScore,GbestPositon,Curve

                            # Set relevant parameters.
time_start = time.time()
pop = 50                    # Flamingo population size.
MaxIter = 300               # Maximum number of iterations.
dim = 20                    # The dimension.
fl=-100                     # The lower bound of the search interval.
ul=100                      # The upper bound of the search interval.
lb = fl*np.ones([dim, 1])
ub = ul*np.ones([dim, 1])
MP_b=0.1
              # The basic proportion of flamingos migration in the first stage.
```

```
GbestScore, GbestPositon, Curve = MSA(pop, dim, lb, ub, MaxIter, fun, MP_b)

time_end = time.time()
y_year_pred = reg.predict(X_year)
y_year_pred1= []
for i in range(0, len(y_year_pred)):
    y_year_pred1.append(y_year_pred[i] * t2[i])
print (np.mean(y_year),np.mean(y_year_pred))
print (np.sqrt(np.var(y_year)),np.sqrt(np.var(y_year_pred)))
plot_graphs(y_year,y_year_pred1,"Prediction in CUTTACK")
print("Prediction in CUTTACK by month wise",y_year_pred1);
if rain<1000:
  print("The selected area -very low rainfall area");
elif rain<2000:
  print("The selected area -low rainfall area");
elif ((rain>=2000) or (rain>=3000)):
  print("The selected area -medium rainfall area");
elif ((rain>=3000) or (rain>=4000)):
  print("The selected area -high rainfall area");
elif rain>4000:
  print("The selected area -very high rainfall area");



# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [245,124,40,4]
prediction = [277,138,63,38]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Long term rainfall-CUTTACK",fontweight ='bold', fontsize = 15)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['Jun', 'JULY', 'AUG', 'SEP'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center',fontsize=20)
plt.show()



# set width of bar
barWidth = 0.25
fig,ax = plt.subplots(figsize =(12, 8))

# set height of bar
Groundtruth= [43,74,227]
prediction = [65,168,114]


# Set position of bar on X axis
br1 = np.arange(len(Groundtruth))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Groundtruth, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Months', fontweight ='bold', fontsize = 25)
plt.title("Short term rainfall-CUTTACK",fontweight ='bold', fontsize = 15)
plt.ylabel('Rainfall Range', fontweight ='bold', fontsize = 25)
plt.xticks([r + 0.5*barWidth for r in range(len(Groundtruth))],
        ['JAN', 'FEB','MAR'],fontweight ='bold', fontsize = 15)
for axis in ['top','bottom','left','right']:
```

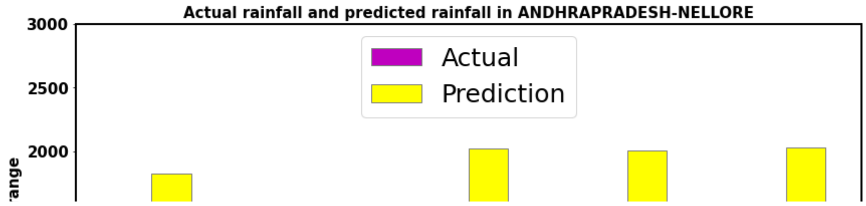
```
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center',fontsize=20)
plt.show()
```

```
from urllib.request import AbstractBasicAuthHandler
barWidth = 0.25
fig,ax = plt.subplots(figsize =(18, 8))

# set height of bar
Actual= [1091,622.8,2647,419.8,1519.7]
prediction=[2028,1457,3133,976,1512]


# Set position of bar on X axis
br1 = np.arange(len(Actual))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Actual, color ='m', width = barWidth,
        edgecolor ='grey', label ='Actual')
plt.bar(br2, prediction, color ='c', width = barWidth,
        edgecolor ='grey', label ='Prediction')
plt.ylim(0,5000)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 25)
plt.title("Comparison-Rainfall",fontweight ='bold', fontsize = 25)
plt.ylabel(' Rainfall types', fontweight ='bold', fontsize = 25)
plt.yticks(np.arange(0, 5000, step=1000),['Very Low RF(<1000)', 'Low RF(1000-2000)', 'Medium RF(2000-3000)','High RF(3000-4000)','Very hi
```

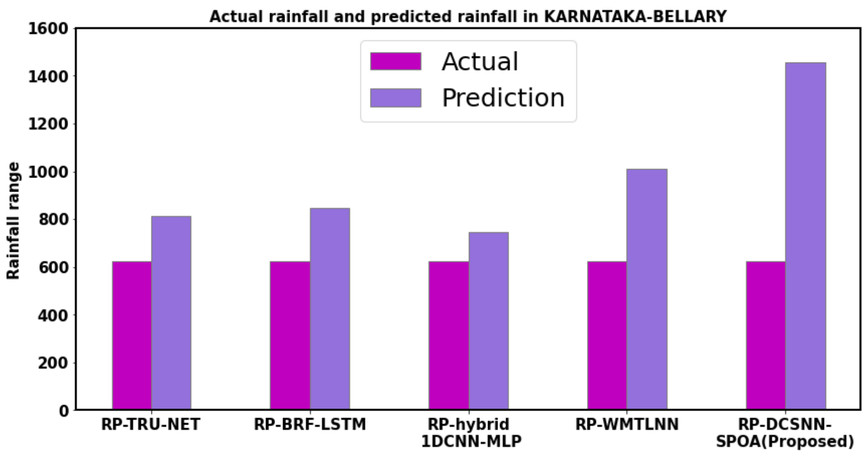
```
plt.xticks([r + 0.4*barWidth for r in range(len(Actual))],
        ['ANDHRAPRADESH \nNELLORE', 'KARNATAKA\nBELLARY', 'MANIPUR\nCHANDEL','PUNJAB\nMANSA','ORRISA\nCUTTACK'], fontsize = 15,fontweight
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center', fontsize = 25)
plt.show()
```



```
barWidth = 0.25
fig,ax = plt.subplots(figsize =(14, 7))

# set height of bar
Actual= [1091,1091,1091,1091,1091]
prediction = [1828,1582,2018,2008,2028]


# Set position of bar on X axis
br1 = np.arange(len(Actual))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1,Actual, color ='m', width = barWidth,
        edgecolor ='grey', label ='Actual')
plt.bar(br2, prediction, color ='yellow', width = barWidth,
        edgecolor ='grey', label ='Prediction')
plt.ylim(0,3000)
plt.yticks(fontsize=15,fontweight='bold')
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 15)
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Actual rainfall and predicted rainfall in ANDHRAPRADESH-NELLORE",fontweight ='bold', fontsize = 15)
#plt.ylabel(' Rainfall types', fontweight ='bold', fontsize = 15)


plt.xticks([r + 0.5*barWidth for r in range(len(Actual))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 15,fontweight ='bold',rc
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center', fontsize = 25)
plt.show()
```

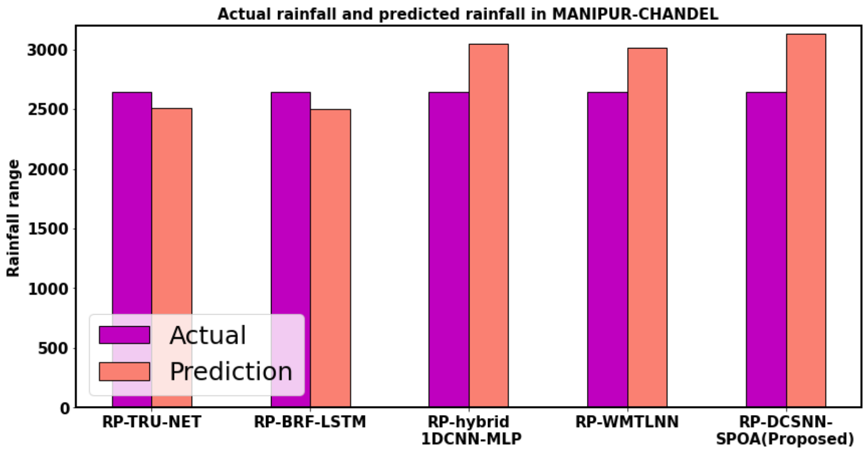



```
barWidth = 0.25
fig,ax= plt.subplots(figsize =(14, 7))

# set height of bar
Actual= [622.8,622.8,622.8,622.8,622.8]
prediction = [813,848,747,1011,1457]


# Set position of bar on X axis
br1 = np.arange(len(Actual))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Actual, color ='m', width = barWidth,
        edgecolor ='grey', label ='Actual')
plt.bar(br2, prediction, color ='mediumpurple', width = barWidth,
        edgecolor ='grey', label ='Prediction')
plt.ylim(0,1600)
plt.yticks(fontsize=15,fontweight='bold')
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 15)
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Actual rainfall and predicted rainfall in KARNATAKA-BELLARY",fontweight ='bold', fontsize = 15)
#plt.ylabel(' Rainfall types', fontweight ='bold', fontsize = 15)


plt.xticks([r + 0.5*barWidth for r in range(len(Actual))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 15,fontweight ='bold',rc
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center', fontsize = 25)
plt.show()
```

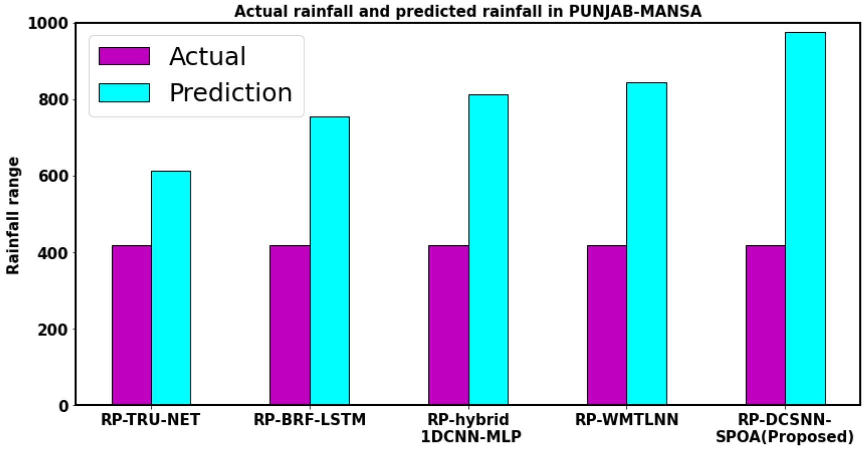



```
barWidth = 0.25
fig, ax = plt.subplots(figsize=(14, 7))

# set height of bar
Actual= [2647.5,2647.5,2647.5,2647.5,2647.5]
prediction = [2513,2498,3047,3011,3133]


# Set position of bar on X axis
br1 = np.arange(len(Actual))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Actual, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
```

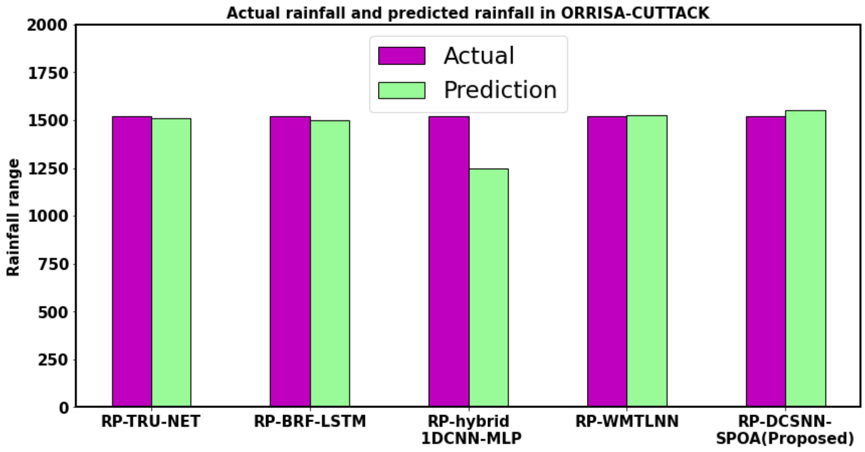
```
plt.bar(br2, prediction, color ='salmon', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.ylim(0,3200)
plt.yticks(fontsize=15,fontweight='bold')
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 15)
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Actual rainfall and predicted rainfall in MANIPUR-CHANDEL",fontweight ='bold', fontsize = 15)
#plt.ylabel(' Rainfall types', fontweight ='bold', fontsize = 15)


plt.xticks([r + 0.5*barWidth for r in range(len(Actual))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 15,fontweight ='bold',rc
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='lower left', fontsize = 25)
plt.show()
```




```
barWidth = 0.25
#fig = plt.subplots(figsize =(14, 7))
fig, ax = plt.subplots(figsize=(14, 7))
# set height of bar
Actual= [419.8,419.8,419.8,419.8,419.8]
prediction = [614,754,812,845,976]


# Set position of bar on X axis
br1 = np.arange(len(Actual))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1,Actual, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='aqua', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.ylim(0,1000)
plt.yticks(fontsize=15,fontweight='bold')
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 15)
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Actual rainfall and predicted rainfall in PUNJAB-MANSA",fontweight ='bold', fontsize = 15)
#plt.ylabel(' Rainfall types', fontweight ='bold', fontsize = 15)


plt.xticks([r + 0.5*barWidth for r in range(len(Actual))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 15,fontweight ='bold',rc
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', fontsize = 25)
plt.show()
```

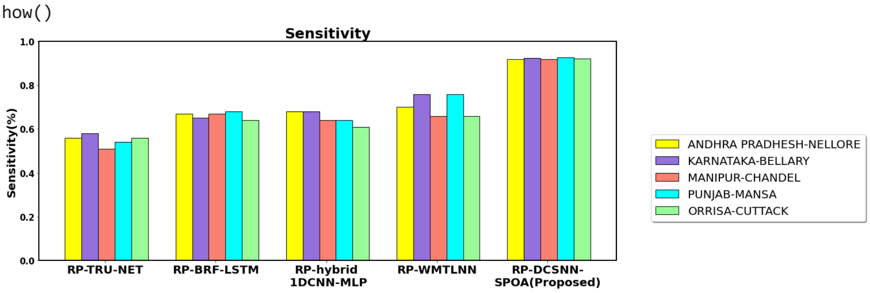



```
barWidth = 0.25
fig ,ax= plt.subplots(figsize =(14, 7))

# set height of bar
Actual= [1519.7,1519.7,1519.7,1519.7,1519.7]
prediction = [1513,1498,1247,1525,1551]


# Set position of bar on X axis
br1 = np.arange(len(Actual))
br2 = [x + barWidth for x in br1]

# Make the plot
plt.bar(br1, Actual, color ='m', width = barWidth,
        edgecolor ='k', label ='Actual')
plt.bar(br2, prediction, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='Prediction')
plt.ylim(0,2000)
plt.yticks(fontsize=15,fontweight='bold')
plt.ylabel('Rainfall range', fontweight ='bold', fontsize = 15)
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Actual rainfall and predicted rainfall in ORRISA-CUTTACK",fontweight ='bold', fontsize = 15)
#plt.ylabel(' Rainfall types', fontweight ='bold', fontsize = 15)


plt.xticks([r + 0.5*barWidth for r in range(len(Actual))],
           ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 15,fontweight ='bold'
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper center', fontsize = 23)
plt.show()
```



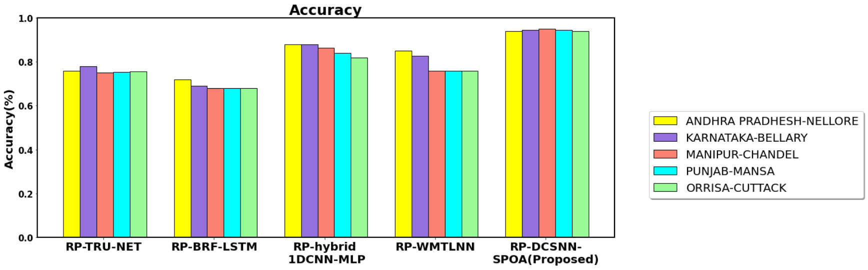

```
barWidth = 0.15
fig ,ax= plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[0.56,0.67,0.68,0.7,.92]
```

```
KARNATAKA_BELLARY=[0.58,0.65,0.68,0.76,.925]
MANIPUR_CHANDEL=[.51,.67,.64,0.66,.92]
PUNJAB_MANSA=[.54,.68,.64,0.76,.926]
ORRISA_CUTTACK=[.56,.64,.61,0.66,.921]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
plt.ylim(0,1)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Sensitivity",fontweight ='bold', fontsize = 25)
plt.ylabel('Sensitivity(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```

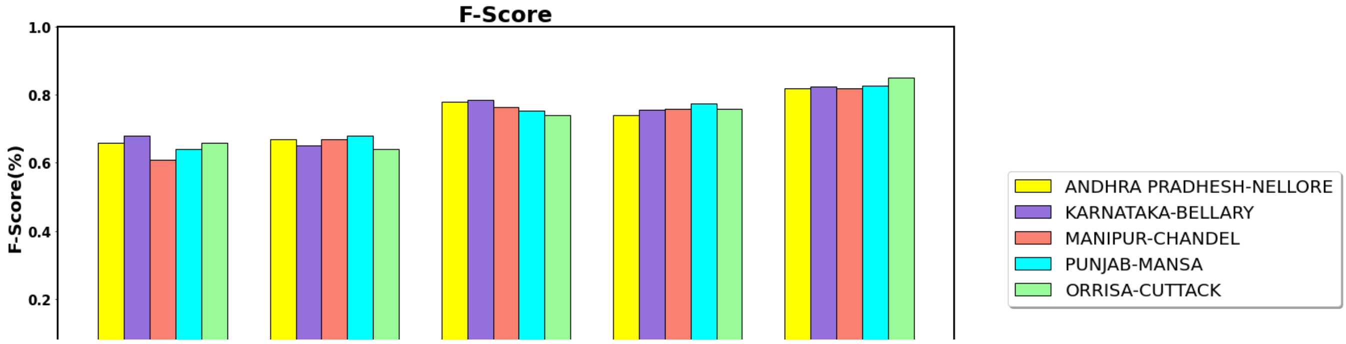



```
barWidth = 0.15
fig,ax = plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[0.76,0.72,0.88,0.85,.94]
KARNATAKA_BELLARY=[0.78,0.69,0.88,0.826,.945]
MANIPUR_CHANDEL=[.751,.68,.864,0.76,.95]
PUNJAB_MANSA=[.754,.68,.84,0.76,.946]
ORRISA_CUTTACK=[.756,.68,.82,0.76,.941]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
```

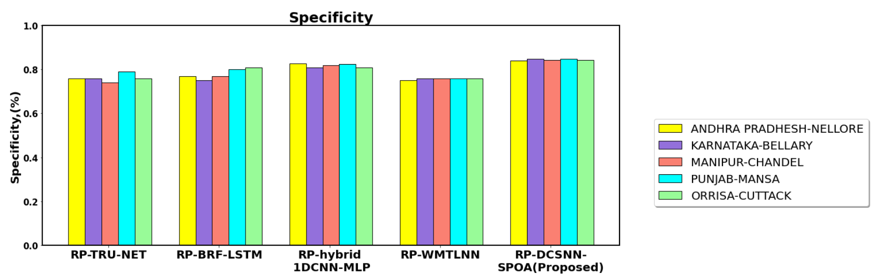
```
plt.ylim(0,1)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Accuracy",fontweight ='bold', fontsize = 25)
plt.ylabel('Accuracy(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```




```
barWidth = 0.15
fig,ax= plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[0.66,0.67,0.78,0.74,.82]
KARNATAKA_BELLARY=[0.68,0.65,0.785,0.756,.825]
MANIPUR_CHANDEL=[.61,.67,.764,0.76,.82]
PUNJAB_MANSA=[.64,.68,.754,0.776,.826]
ORRISA_CUTTACK=[.66,.64,.74,0.76,.851]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
plt.ylim(0,1)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("F-Score",fontweight ='bold', fontsize = 25)
plt.ylabel('F-Score(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```

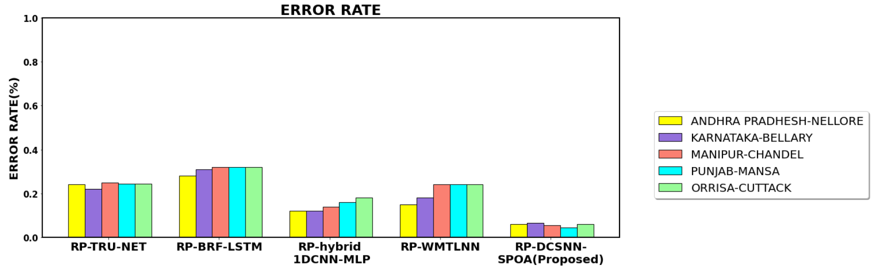



```
barWidth = 0.15
fig,ax= plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[0.76,0.77,0.828,0.75,.840]
KARNATAKA_BELLARY=[0.758,0.75,0.81,0.76,.849]
MANIPUR_CHANDEL=[.74,.77,.82,0.76,.842]
PUNJAB_MANSA=[.79,.80,.824,0.76,.8476]
ORRISA_CUTTACK=[.76,.81,.81,0.76,.8421]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
plt.ylim(0,1)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Specificity",fontweight ='bold', fontsize = 25)
plt.ylabel('Specificity,(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```



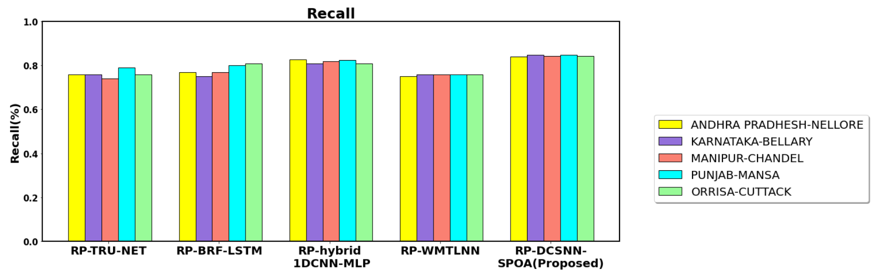

```
barWidth = 0.15
fig,ax = plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[.24,.28,.12,.15,.06]
KARNATAKA_BELLARY=[.22,.31,.12,.18,.065]
```

```
MANIPUR_CHANDEL=[.25,.32,.14,.24,.054]
PUNJAB_MANSA=[.245,.32,.16,.24,.045]
ORRISA_CUTTACK=[.245,.32,.18,.24,.06]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
plt.ylim(0,1)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("ERROR RATE",fontweight ='bold', fontsize = 25)
plt.ylabel('ERROR RATE(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```




```
barWidth = 0.15
fig ,ax= plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[0.76,0.77,0.828,0.75,.840]
KARNATAKA_BELLARY=[0.758,0.75,0.81,0.76,.849]
MANIPUR_CHANDEL=[.74,.77,.82,0.76,.842]
PUNJAB_MANSA=[.79,.80,.824,0.76,.8476]
ORRISA_CUTTACK=[.76,.81,.81,0.76,.8421]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
plt.ylim(0,1)
```

```
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Recall",fontweight ='bold', fontsize = 25)
plt.ylabel('Recall(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```




```
barWidth = 0.15
fig,ax = plt.subplots(figsize =(18, 7))

# set height of bar
ANDHRAPRADHESH_NELLORE=[0.56,0.69,0.68,0.7,.912]
KARNATAKA_BELLARY=[0.58,0.65,0.68,0.86,.925]
MANIPUR_CHANDEL=[.55,.66,.64,0.86,.912]
PUNJAB_MANSA=[.54,.68,.64,0.76,.926]
ORRISA_CUTTACK=[.56,.64,.61,0.86,.921]
# Set position of bar on X axis
br1 = np.arange(len(ANDHRAPRADHESH_NELLORE))
br2 = [x + barWidth for x in br1]
br3 = [x + barWidth for x in br2]
br4 = [x + barWidth for x in br3]
br5 = [x + barWidth for x in br4]
# Make the plot
plt.bar(br1, ANDHRAPRADHESH_NELLORE, color ='yellow', width = barWidth,
        edgecolor ='k', label ='ANDHRA PRADHESH-NELLORE')
plt.bar(br2, KARNATAKA_BELLARY, color ='mediumpurple', width = barWidth,
        edgecolor ='k', label ='KARNATAKA-BELLARY')
plt.bar(br3, MANIPUR_CHANDEL, color ='salmon', width = barWidth,
        edgecolor ='k', label ='MANIPUR-CHANDEL')
plt.bar(br4, PUNJAB_MANSA, color ='aqua', width = barWidth,
        edgecolor ='k', label ='PUNJAB-MANSA')
plt.bar(br5, ORRISA_CUTTACK, color ='palegreen', width = barWidth,
        edgecolor ='k', label ='ORRISA-CUTTACK')
plt.ylim(0,1)
plt.yticks(fontsize=15,fontweight='bold')
# Adding Xticks
#plt.xlabel('Different Area in INDIA', fontweight ='bold', fontsize = 15)
plt.title("Precision",fontweight ='bold', fontsize = 25)
plt.ylabel('Precision(%)', fontweight ='bold', fontsize = 20)


plt.xticks([r + 1.92*barWidth for r in range(len(ANDHRAPRADHESH_NELLORE))],
        ['RP-TRU-NET','RP-BRF-LSTM','RP-hybrid\n 1DCNN-MLP','RP-WMTLNN','RP-DCSNN-\nSPOA(Proposed)'], fontsize = 20,fontweight="bold",rot
for axis in ['top','bottom','left','right']:
    ax.spines[axis].set_linewidth(2)
plt.legend(loc='upper left', bbox_to_anchor = (1.05, 0.6),
          ncol=1, fancybox=True, shadow=True,fontsize=20)
plt.show()
```