LOGIN OR HOMPAGE MODULE

HTML,CSS CODE:

```
<!DOCTYPE html>
<html>
<head>
   <title>Login Form</title>
   <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
   <h2>Login Page</h2><br>
   <div class="login">
   <form id="login" method="get" action="login.php">
      <label><b>User Name
      </b>
      </label>
      <input type="text" name="Uname" id="Uname" placeholder="Username">
      <br><br>
      <label><b>Password
      </b>
      </label>
      <input type="Password" name="Pass" id="Pass" placeholder="Password">
      <br><br>
      <input type="button" name="log" id="log" value="Log In Here">
      <br><br>
      <input type="checkbox" id="check">
      <span>Remember me</span>
      <br><br>
      <a href="#">Forgot Password</a>
   </form>
</div>
</body>
</html>
body
{
   margin: 0;
   padding: 0;
   background: url("bg.jpeg");
   background-repeat: no-repeat;
   background-size: cover;
   font-family: 'Arial';
}
.login{
      width: 382px;
      overflow: hidden;
```

```
        margin: auto;
        margin: 20 0 0 450px;
        padding: 80px;
        background: #23463f;
        border-radius: 15px ;

}
h2{
    text-align: center;
    color: #1f423b;
    padding: 20px;
}
label{
    color: #08ffd1;
    font-size: 17px;
}
#Uname{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 3px;
    padding-left: 8px;
}
#Pass{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 3px;
    padding-left: 8px;

}
#log{
    width: 300px;
    height: 30px;
    border: none;
    border-radius: 17px;
    padding-left: 7px;
    color: blue;


}
span{
    color: white;
    font-size: 17px;
}
a{
    float: left;
    color: white;
    background-color: #23463f;
}
```

FOR RAINFALL AND FLOOD PREDICTION:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
#import mpld3

# ## Types of graphs
# - Bar graphs showing distribution of amount of rainfall.
# - Distribution of amount of rainfall yearly, monthly, groups of months.
# - Distribution of rainfall in subdivisions, districts form each month, groups of months.
# - Heat maps showing correlation between amount of rainfall between months.
#

# In[2]:
def rainfall(year,region):
    data = pd.read_csv(‘data/Sub_Division_IMD_2017.csv’)
    data = data.fillna(data.mean())
    data.info()


    # In[3]:

    data.head()


    # In[4]:

    data.describe()


    # In[5]:

    # data.hist(figsize=(12,12));


    # ## Observations
    # - Above histograms show the distribution of rainfall over months.
    # - Observed increase in amount of rainfall over months July, August, September.
    # In[6]:

    # data[[‘YEAR’, ‘JAN’, ‘FEB’, ‘MAR’, ‘APR’, ‘MAY’, ‘JUN’, ‘JUL’,
    #       ‘AUG’, ‘SEP’, ‘OCT’, ‘NOV’, ‘DEC’]].groupby(“YEAR”).sum().plot(figsize=(13,8));


    # In[7]:

    # data[[‘YEAR’,’JF’, ‘MAM’,
    #       ‘JJAS’, ‘OND’]].groupby(“YEAR”).sum().plot(figsize=(13,8));


    # ## Observations
    # - The above two graphs show the distribution of rainfall over months.
    # - The graphs clearly shows that amount of rainfall in high in the months    oda, aug, sep which is
monsoon season in India.

    # In[8]:

    # data[[‘SUBDIVISION’, ‘JAN’, ‘FEB’, ‘MAR’, ‘APR’, ‘MAY’, ‘JUN’, ‘JUL’,
    #       ‘AUG’, ‘SEP’, ‘OCT’, ‘NOV’,
‘DEC’]].groupby(“SUBDIVISION”).mean().plot.barh(stacked=True,figsize=(13,8));


    # In[9]:

    # data[[‘SUBDIVISION’, ‘JF’, ‘MAM’,
    #       ‘JJAS’, ‘OND’]].groupby(“SUBDIVISION”).sum().plot.barh(stacked=True,figsize=(16,8));


    # In[10]:

    # plt.figure(figsize=(11,4))
    # sns.heatmap(data[[‘JF’, ‘MAM’,
    #       ‘JJAS’, ‘OND’,’ANNUAL’]].corr(),annot=True)
    # plt.show()


    # In[11]:

    # plt.figure(figsize=(11,4))
    #
sns.heatmap(data[[‘JAN’,’FEB’,’MAR’,’APR’,’MAY’,’JUN’,’JUL’,’AUG’,’SEP’,’OCT’,’NOV’,’DE
C’,’ANNUAL’]].corr(),annot=True)
    # plt.show()
    # ## Observations
    # - **Heat Map** shows the co-relation(dependency)    odavar the amounts of rainfall over months.
    # - From above it is clear that if amount of rainfall is high in the months of    oda, august,
    odavari    then the amount of rainfall will be high annually.
    # - It is also obwserved that if amount of rainfall in good in the months of    odavar,    odavari,
    odavari then the rainfall is going to b good in the overall year.

    # In[17]:

    #Function to plot the graphs
    def plot_graphs(groundtruth,prediction,title):
        N = 9
        ind = np.arange(N)  # the x locations for the groups
        width = 0.27       # the width of the bars

        fig = plt.figure(figsize=(18,10))
        fig.suptitle(title, fontsize=12)
        ax = fig.add_subplot(111)
        rects1 = ax.bar(ind, groundtruth, width, color=’m’)
        rects2 = ax.bar(ind+width, prediction, width, color=’c’)

        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’) )

    #     autolabel(rects1)
        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’)
    #     autolabel(rects2)

        #plt.show()
        #mpld3.save_html(fig,’static/img/rainfall.html’)
        plt.savefig(‘static/img/rainfall.png’)


    def data_generation(year,region):
        temp = data[[‘SUBDIVISION’,’JAN’, ‘FEB’, ‘MAR’, ‘APR’, ‘MAY’, ‘JUN’, ‘JUL’,
            ‘AUG’, ‘SEP’, ‘OCT’, ‘NOV’, ‘DEC’]].loc[data[‘YEAR’] == year]
        data_year = np.asarray(temp[[‘JAN’, ‘FEB’, ‘MAR’, ‘APR’, ‘MAY’, ‘JUN’, ‘JUL’,
            ‘AUG’, ‘SEP’, ‘OCT’, ‘NOV’, ‘DEC’]].loc[temp[‘SUBDIVISION’] == region])
        X_year = None; y_year = None
        for I in range(data_year.shape[1]-3):
            if X_year is None:
                X_year = data_year[:, i:i+3]
                y_year = data_year[:, i+3]
            else:
                X_year = np.concatenate((X_year, data_year[:, i:i+3]), axis=0)
                y_year = np.concatenate((y_year, data_year[:, i+3]), axis=0)

        return X_year,y_year

    def data_generation2(region):
        Kerala = np.asarray(data[[‘JAN’, ‘FEB’, ‘MAR’, ‘APR’, ‘MAY’, ‘JUN’, ‘JUL’,
              ‘AUG’, ‘SEP’, ‘OCT’, ‘NOV’, ‘DEC’]].loc[data[‘SUBDIVISION’] == region])

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



        return X,y

    def prediction2(year,region):
        from keras.models import Model
        from keras.layers import Dense, Input, Conv1D, Flatten

        # NN 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’])
        X_testing,Y_testing = data_generation(year,region)
```

```
        from sklearn.metrics import mean_absolute_error
        from sklearn.metrics import explained_variance_score
        # linear model

        X_train,y_train = data_generation2(region)
        model.fit(x=np.expand_dims(X_train, axis=2), y=y_train, batch_size=64, epochs=20, verbose=1,
validation_split=0.1, shuffle=True)

        y_pred = model.predict(np.expand_dims(X_testing, axis=2))
        mae=mean_absolute_error(Y_testing, y_pred)
        score=explained_variance_score(Y_testing, y_pred)
        print(mae)
        print(score)

        Y_year_pred=list(range(9))
        for I in range(9):
           Y_year_pred[i]=y_pred[i][0]
        y_pred=np.array(Y_year_pred)
        plot_graphs(Y_testing,y_pred,"Year: "+str(year)+'  Region: '+str(region))
        return mae,score

    print("###########",year,type(int(year)),region,type(region),"7777777777777777777777777&&
    &&&&&&&&&&&&&&&&&&&&&&&&&")
     mae,score=prediction2(int(year),region)
     mae=format(round(float(mae),2))
     score=format(round(float(score),2))
     keras.backend.clear_session()
     return mae,score
     import pandas as pd
from datetime import datetime
import time
import matplotlib.pyplot as plt
from sklearn.preprocessing import Normalizer,MinMaxScaler
from imblearn.over_sampling import SMOTE
from sklearn.utils import shuffle
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import classification_report
import numpy as np
import seaborn as sns
import plotly.graph_objs as go
import plotly.plotly as py
import plotly
from sklearn.externals import joblib

import warnings
warnings.filterwarnings("ignore")
```

```
#get_ipython().run_line_magic('matplotlib', 'inline')
#fd-future data set
#validating-0 or 1 (0-tetsing ,1= future prediction)
def flood_classifier(filename,fd,validating=0):

    data1=pd.read_excel('data/'+filename+'.xlsx')

    # In[4]:
    data1.shape
    # In[5]:

    #Fillng null entries with mean of their respective columns
    for I in range(1,len(data1.columns)):
        data1[data1.columns[i]] = data1[data1.columns[i]].fillna(data1[data1.columns[i]].mean())
    # In[6]:
    data1.describe()
    # In[7]:
    y=data1['Flood']
    # In[8]:
    for I in range(len(y)):
        if(y[i] >= 0.1):
            y[i]=1
    # In[9]:

    y=pd.DataFrame(y)

    data1.drop('Flood',axis=1,inplace=True)


    # In[10]:
    data1.head()
    # In[11]:
    data1.hist(figsize=(6,6));

    #Breaking Date column into timestamp

    d1=pd.DataFrame()
    d1["Day"]=data1['Date']
    d1['Months']=data1['Date']
    d1['Year']=data1['Date']
    data1['Date']=pd.to_datetime(data1['Date'])
    d1["Year"]=data1.Date.dt.year
    d1["Months"]=data1.Date.dt.month
    d1["Day"]=data1.Date.dt.day

    #---------------------Resampling
```

```
#------------not working for    odava
dx=pd.DataFrame()
dx[‘Date’]=data1[‘Date’]
dx[‘Discharge’]=data1[‘Discharge’]
dx=dx.set_index([‘Date’])
yearly = dx.resample(‘Y’).sum()

plt.figure(figsize=(9,8))
plt.xlabel(‘YEARS’)
plt.ylabel(‘Level’)
plt.title(filename+” : Year wise Trends”)
plt.plot(yearly,’—')

#plt.plot(yearly,style=[‘:’, ‘—', ‘-‘],title=’Year wise Trends’)
plt.savefig(‘static/img/flood.png’)
#--------------------------------



# In[18]:
data1.drop(‘Date’,inplace=True,axis=1)
# In[19]:



#Scaling the data in range of 0 to 1

# Scaler=MinMaxScaler(feature_range=(0, 1))
# Transform=Scaler.fit_transform(data1)
# # In[20]
# #Transform
# # In[21]:
# Transform=pd.DataFrame(Transform,columns=[‘Discharge’,’flood runoff’,’daily runoff’,’weekly
runoff’])

# # In[22]:
# data1=Transform
# In[23]:
data1=pd.concat([d1,data1],axis=1)
data1.head()

#----------------------for taking data upto 2015 as training and rest for testing------------------------------
------------------
locate=0;
for I in range(len(data1[“Day”])):
    if(data1[“Day”][i]==31 and data1[“Months”][i]==12 and data1[“Year”][i]==2015):
        locate=I;
        break;
```

```
i=locate+1
print(i)

x_train=data1.iloc[0:I,:]
y_train=y.iloc[0:i]
x_test=data1.iloc[i:,:]
y_test=y.iloc[i:]


# In[25]:


x_train.drop(labels=['Day','Months','Year'],inplace=True,axis=1)
x_test.drop(labels=['Day','Months','Year'],inplace=True,axis=1)


# In[26]:


# nl=Normalizer()
# x_train=nl.fit_transform(x_train)
# x_test=nl.transform(x_test)


# In[27]:


# y_train=nl.transform(y_train)
# y_test=nl.transform(y_test)


# In[28]:


#-----------------Upsampling the data (as very less entries of flood =1 is present)-----------------
sm = SMOTE(random_state=2)
X_train_res, Y_train_res = sm.fit_sample(x_train, y_train)
# In[29]:
X_train_res.shape
# In[30]:
x_train, y_train = shuffle( X_train_res, Y_train_res, random_state=0)

x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
# In[32]:


# #---------------Logistic Regression-------------------------
# from sklearn.linear_model import LogisticRegression
# reg=LogisticRegression()
# reg.fit(x_train,y_train)
# y_predict1=reg.predict(x_test)
# print(set(y_predict1))
# print(reg.score(x_train,y_train))
# print(reg.score(x_test,y_test))
# print(classification_report(y_test, y_predict1))
# print("mean_absolute_error=",mean_absolute_error(y_test, y_predict1))


# # In[34]:

#----------------------LinearDiscriminantAnalysis--------------------------------
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# clf1=LinearDiscriminantAnalysis()
# clf1.fit(x_train,y_train)

#----------------------saving & Loading the model-----------------------------------------

path='trained/'+filename+'_LDA'
#joblib.dump(clf1, path+'.pkl')
clf1= joblib.load(path+'.pkl')

#---------------------------------------------------------------------------

y_predict3=clf1.predict(x_test)
print(set(y_predict3))
print(clf1.score(x_train,y_train))
print(clf1.score(x_test,y_test))
print(classification_report(y_test, y_predict3))
mae=mean_absolute_error(y_test, y_predict3)
print("mean_absolute_error=",mae)

# In[35]:
#--------------------------KneighborsClassifier----------------------------------------

# from sklearn.neighbors import KneighborsClassifier

# clf2=KneighborsClassifier()
# clf2.fit(x_train,y_train)
```

```
   # y_predict4=clf2.predict(x_test)
   # print(set(y_predict4))
   # print(clf2.score(x_train,y_train))
   # print(clf2.score(x_test,y_test))
   # print(classification_report(y_test, y_predict4))
   # print(“mean_absolute_error=”,mean_absolute_error(y_test, y_predict4))

   # # In[36]:

   #-------------------------------Testing----------------------------------------------
   # In[38]:
   data1.head()
   # In[39]:
   def predicting(future_data):
           # xx=[13214.0,0.0,0.36,2.08]
           #xx=[4990.0,0.0,1.40,15.38]
           xx=future_data
           xx=np.array(xx)
           xx=xx.reshape((-1, 4))
           xx=clf1.predict(xx)
           # xx=reg.predict(xx)
           return xx
   xx=predicting(fd)
   return xx,mae
#xx=predicted value of flood 0 or 1

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from datetime import datetime
#data=pd.read_csv(“   odavari_daily.csv”)
data=pd.read_excel(“data/Godavari.xlsx”)
y=data[‘Flood’]
# data.drop(‘Flood’,axis=1,inplace=True)
set(y)
y.plot(x = data[‘Date’], y = y)
for I in range(len(y)):
    if y.iloc[i] >= 0.5 :
        y.iloc[i]=1.0
    else :
        y.iloc[i]=0.0
data.drop(‘Flood’,axis=1,inplace=True)
d1=pd.DataFrame()
d1[“Day”]=data[‘Date’]
```

```
d1[‘Months’]=data[‘Date’]
d1[‘Year’]=data[‘Date’]
d1[“Year”]=data.Date.dt.year
d1[“Months”]=data.Date.dt.month
d1[“Day”]=data.Date.dt.day
d1.head()
x=pd.DataFrame(data=data[‘Day’])
x[‘Months’]=data[‘Months’]
x[‘Year’]=data[‘Year’]
x.head()
data1=pd.concat([x,data1],axis=1)
data=data1
data.head()
data[[‘Year’, ‘flood runoff’, ‘daily runoff’, ‘weekly
runoff’]].groupby(“Year”).sum().plot(figsize=(13,8));
```

TRAINING RAINFALL PREDICTION MODEL WITH DIFFERENT MODELS :

We will divide the dataset into training (75%) and test (25%) sets respectively to train the rainfall prediction model. For best results, we will standardize our X_train and X_test data:

CODE-

```
features = MiceImputed[['Location', 'MinTemp', 'MaxTemp', 'Rainfall', 'Evaporation', 'Sunshine',
'WindGustDir',
                       'WindGustSpeed', 'WindDir9am', 'WindDir3pm', 'WindSpeed9am', 'WindSpeed3pm',
'Humidity9am',
                       'Humidity3pm', 'Pressure9am', 'Pressure3pm', 'Cloud9am', 'Cloud3pm', 'Temp9am',
'Temp3pm',
                       'RainToday']]
target = MiceImputed['RainTomorrow']

# Split into test and train
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.25, random_state=12345)

# Normalize Features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
def plot_roc_cur(fper, tper):
    plt.plot(fper, tper, color='orange', label='ROC')
    plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
import time
from sklearn.metrics import accuracy_score, roc_auc_score, cohen_kappa_score,
plot_confusion_matrix, roc_curve, classification_report
def run_model(model, X_train, y_train, X_test, y_test, verbose=True):
    t0=time.time()
    if verbose == False:
        model.fit(X_train,y_train, verbose=0)
    else:
        model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred)
    coh_kap = cohen_kappa_score(y_test, y_pred)
    time_taken = time.time()-t0
    print("Accuracy = {}".format(accuracy))
    print("ROC Area under Curve = {}".format(roc_auc))
    print("Cohen's Kappa = {}".format(coh_kap))
    print("Time taken = {}".format(time_taken))
    print(classification_report(y_test,y_pred,digits=5))

    probs = model.predict_proba(X_test)
    probs = probs[:, 1]
    fper, tper, thresholds = roc_curve(y_test, probs)
    plot_roc_cur(fper, tper)

    plot_confusion_matrix(model, X_test, y_test,cmap=plt.cm.Blues, normalize = 'all')

    return model, accuracy, roc_auc, coh_kap, time_taken
```

RAINFALL PREDICTION MODEL COMPARISON

METRICS FOR ALGORITHMIC COMPARISON :

The implementation of the different machine learning algorithms such as SVM, Bayes, KNN and deep neural network are compared to identify the best algorithm for the prediction of the occurrence of flood. Before the results, a brief explanation about the various metrics used for comparative analysis is given below.

Confusion matrix

This is a binary classifier. A confusion matrix can be of any size depending upon the different number of parameters inputted (labels in our case). The confusion matrix in our case is a 2 × 2 matrix. where TP = true positive; FN = false negative; FP = false positive; TN = true negative. TP and TN denote the number of instances which have been correctly classified as no flood occurrence and flood occurrence respectively. FP and FN signify the number of instances which have been wrongly classified as no flood occurrence and flood occurrence respectively.

Precision and recall

The success of prediction is computed by means of precision-recall where classes are imbalanced. The relevancy of the result is expressed as precision whereas the number of true relevant results returned is expressed as recall. A low false positive rate and low false negative rate result is related to high precision and high recall respectively.

Precision: Termed as the positive predictive value is calculated as given below

$$Precision = \frac{TP}{TP + FP}$$

Recall: Also called as sensitivity is calculated as given below.

$$Recall = \frac{TP}{TP + FN}$$

Accuracy

Based on the confusion matrix created, accuracy computed as an accuracy score function which is either by default the fraction or the count (normalize = false) of correct predictions. Accuracy is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN}$$

F1 score

This is a measure of the test's accuracy where it considers both precision and recall. This is the harmonic average of precision and recall where F1 reaches its best value at 1 and worst at 0. The F1 score conveys the balance between precision and recall:

$$F1\ score = 2 * \frac{\mathrm{Precision} * \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

MCC

Matthews correlation coefficient considers all four divisions of the confusion matrix when calculated. MCC lies within the range −1 to +1 where a model with a positive score is considered to be perfect whereas the negative score is poor. This makes this metric really useful as it is easy to interpret (Gereon 2018):

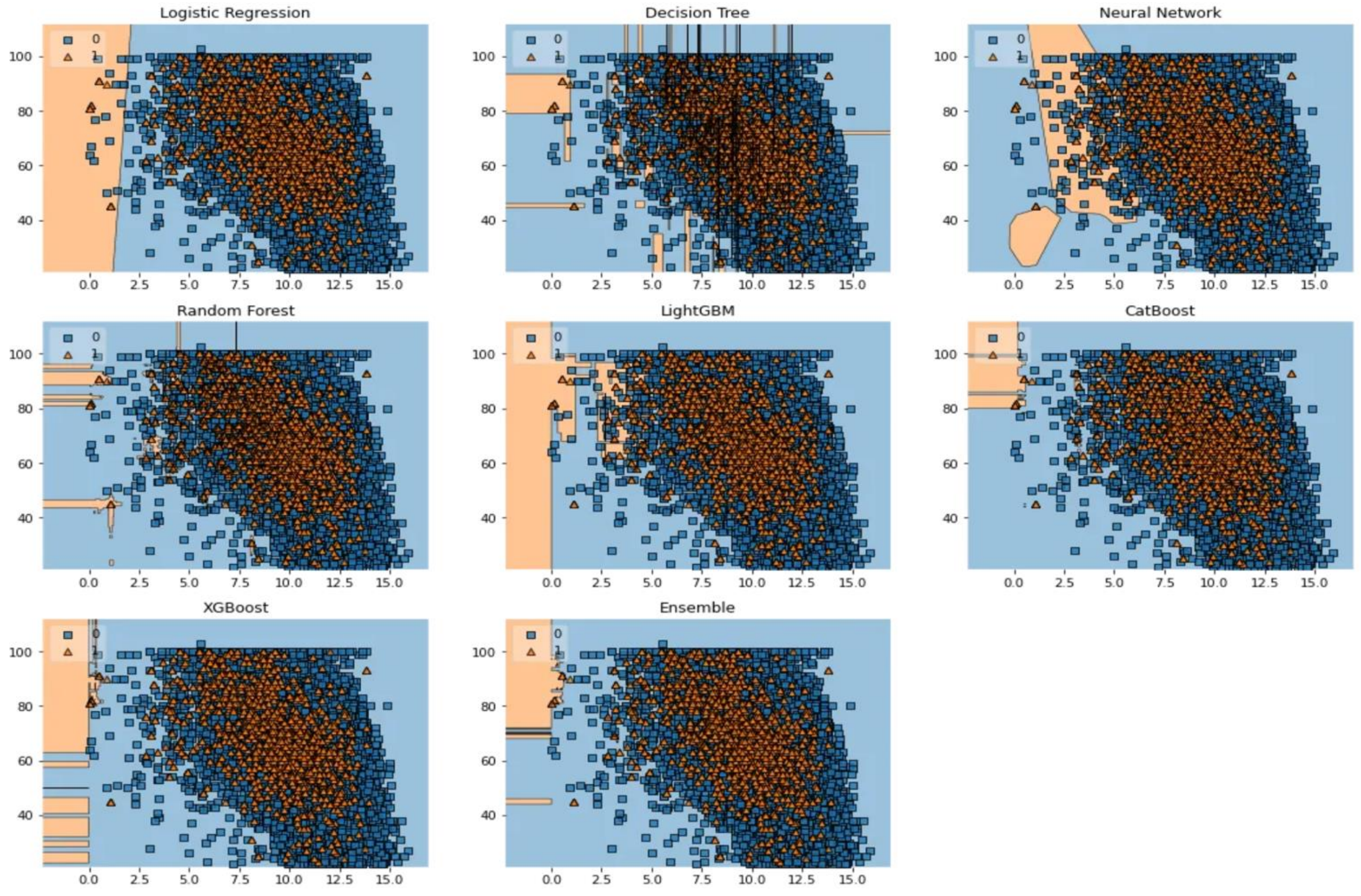
$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}$$

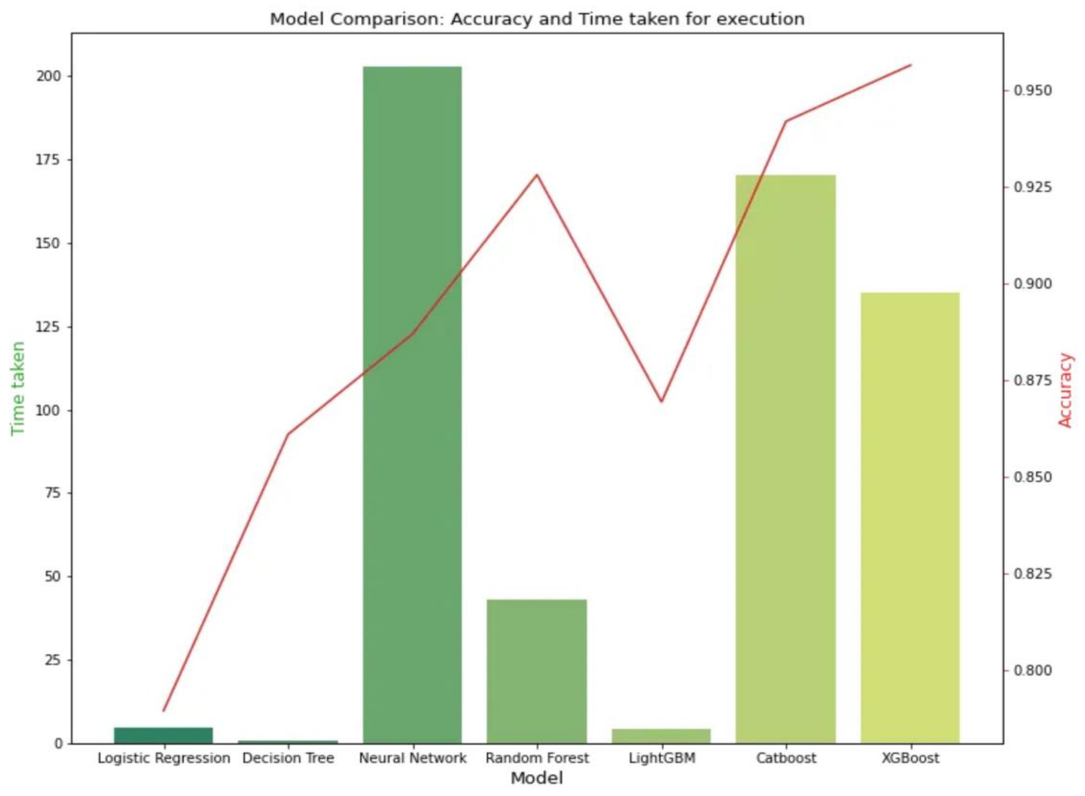
CODE-

```
Accuracy_scores = [accuracy_lr, accuracy_dt, accuracy_nn, accuracy_rf, accuracy_lgb, accuracy_cb,
accuracy_xgb]
roc_auc_scores = [roc_auc_lr, roc_auc_dt, roc_auc_nn, roc_auc_rf, roc_auc_lgb, roc_auc_cb,
roc_auc_xgb]
coh_kap_scores = [coh_kap_lr, coh_kap_dt, coh_kap_nn, coh_kap_rf, coh_kap_lgb, coh_kap_cb,
coh_kap_xgb]
tt = [tt_lr, tt_dt, tt_nn, tt_rf, tt_lgb, tt_cb, tt_xgb]

model_data = {'Model': ['Logistic Regression','Decision Tree','Neural Network','Random
Forest','LightGBM','Catboost','XGBoost'],
            'Accuracy': accuracy_scores,
            'ROC_AUC': roc_auc_scores,
            'Cohen_Kappa': coh_kap_scores,
            'Time taken': tt}
data = pd.DataFrame(model_data)

fig, ax1 = plt.subplots(figsize=(12,10))
ax1.set_title('Model Comparison: Accuracy and Time taken for execution', fontsize=13)
color = 'tab:green'
ax1.set_xlabel('Model', fontsize=13)
ax1.set_ylabel('Time taken', fontsize=13, color=color)
ax2 = sns.barplot(x='Model', y='Time taken', data = data, palette='summer')
ax1.tick_params(axis='y')
ax2 = ax1.twinx()
color = 'tab:red'
ax2.set_ylabel('Accuracy', fontsize=13, color=color)
ax2 = sns.lineplot(x='Model', y='Accuracy', data = data, sort=False, color=color)
ax2.tick_params(axis='y', color=color)
Code language: PHP (php)
  figsize=(12,10))
ax3.set_title('Model Comparison: Area under ROC and Cohens Kappa', fontsize=13)
color = 'tab:blue'
ax3.set_xlabel('Model', fontsize=13)
ax3.set_ylabel('ROC_AUC', fontsize=13, color=color)
ax4 = sns.barplot(x='Model', y='ROC_AUC', data = data, palette='winter')
ax3.tick_params(axis='y')
```

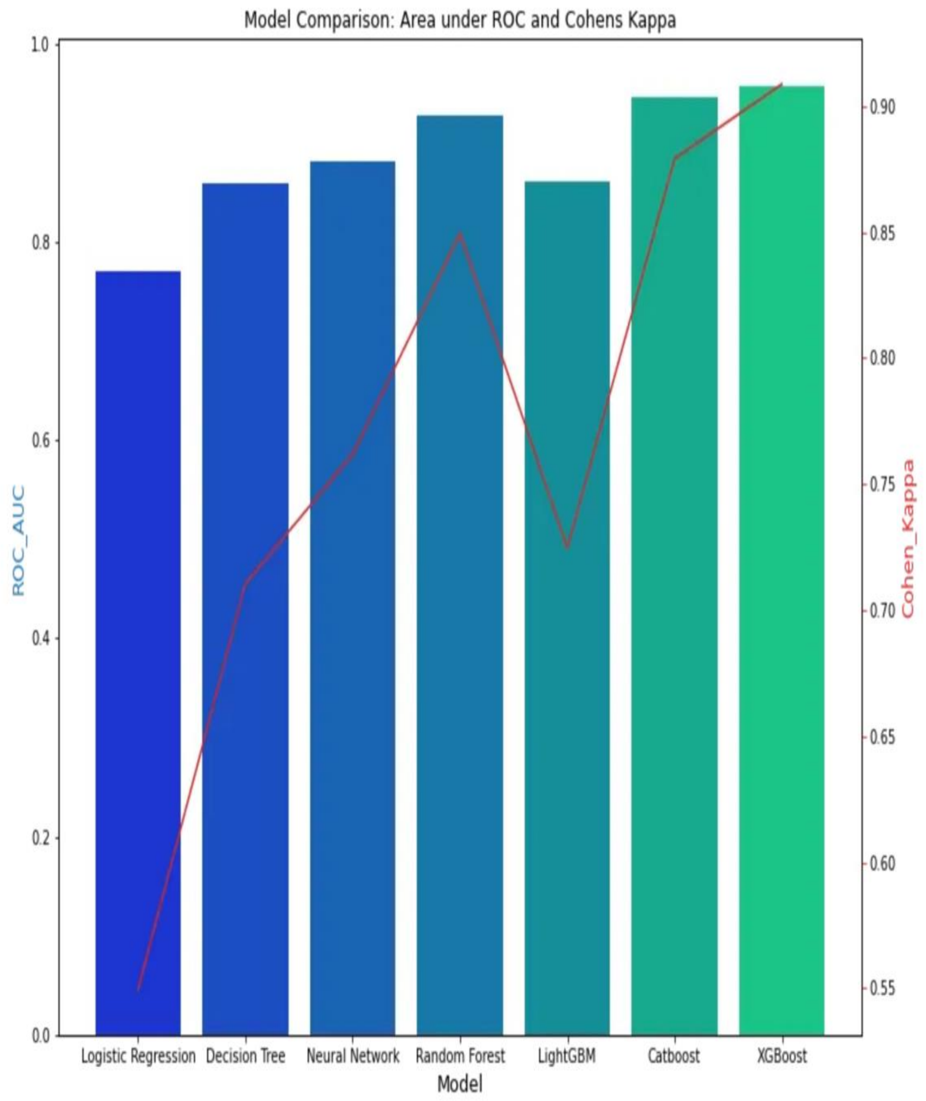
```
ax4 = ax3.twinx()
color = 'tab:red'
ax4.set_ylabel('Cohen_Kappa', fontsize=13, color=color)
ax4 = sns.lineplot(x='Model', y='Cohen_Kappa', data = data, sort=False, color=color)
ax4.tick_params(axis='y', color=color)
plt.show()
```

RAINFALL PREDICTION MODEL COMPARISON:





Model Comparison: Accuracy and Time taken for execution
Time taken
Accuracy
Model
0
25
50
75
100
125
150
175
200
0.800
0.825
0.850
0.875
0.900
0.925
0.950
Logistic Regression
Decision Tree
Neural Network
Random Forest
LightGBM
Catboost
XGBoost



Model Comparison: Area under ROC and Cohens Kappa
ROC_AUC
Cohen_Kappa
Model
0.0
0.2
0.4
0.6
0.8
1.0
0.55
0.60
0.65
0.70
0.75
0.80
0.85
0.90
Logistic Regression
Decision Tree
Neural Network
Random Forest
LightGBM
Catboost
XGBoost