**SOURCE CODE**

**GeneratePltGraph.py**

**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns; sns.set()  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.metrics **import** mean\_absolute\_error  
**from** sklearn.preprocessing **import** StandardScaler  
**import** numpy **as** np *# linear algebra***import** pandas **as** pd *# data processing, CSV file I/O (e.g. pd.read\_csv)***from** sklearn **import** metrics  
**import** os  
**import** matplotlib  
matplotlib.use(**'TKAgg'**)  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
  
  
**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()  
 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'**))  
  
 *# 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)  
 # os.path.join(BASE\_DIR, 'assets/templates')* plt.savefig(os.path.join(BASE\_DIR, **'assets/static/graphs/'**) + title + **"."** + **'png'**)  
 plt.show()  
 plt.close()  
  
  
**class** GeneratePltGraph:  
  
 **def** preProcessGraphs(self,data):  
 data = data.fillna(data.mean())  
 data.info()  
 print(data.head())  
 print(data.describe())  
 *# plt.savefig('models\_accuracy.png') #for Saving pic  
 # plt.savefig('figure1.png',data.hist(figsize=(24, 24)))* data.hist(figsize=(24, 24));  
 *#plt.savefig('hist.png')* data.groupby(**"YEAR"**).sum()[**'ANNUAL'**].plot(figsize=(12, 8));  
 data[[**'YEAR'**, **'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**, **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].groupby(  
 **"YEAR"**).sum().plot(figsize=(13, 8));  
 data[[**'YEAR'**, **'JanToFeb'**, **'MarToMay'**, **'JunToSep'**, **'OctToDec'**]].groupby(**"YEAR"**).sum().plot(figsize=(13, 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));  
 data[[**'SUBDIVISION'**, **'JanToFeb'**, **'MarToMay'**, **'JunToSep'**, **'OctToDec'**]].groupby(**"SUBDIVISION"**).sum().plot.barh(  
 stacked=**True**, figsize=(16, 8));  
 plt.figure(figsize=(11, 4))  
 sns.heatmap(data[[**'JanToFeb'**, **'MarToMay'**, **'JunToSep'**, **'OctToDec'**, **'ANNUAL'**]].corr(), annot=**True**)  
 plt.show()  
  
 plt.figure(figsize=(11, 4))  
 sns.heatmap(  
 data[[**'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**, **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**, **'ANNUAL'**]].corr(),  
 annot=**True**)  
 plt.show()  
 plt.close()  
  
  
  
 **def** genMlrCodes(self,data):  
 *# seperation of training and testing data* 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=0.1, random\_state=42)  
 *# test 2010* temp = data[[**'SUBDIVISION'**, **'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**,  
 **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].loc[data[**'YEAR'**] == 2010]  
  
 data\_2010 = np.asarray(temp[[**'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**,  
 **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].loc[temp[**'SUBDIVISION'**] == **'TELANGANA'**])  
  
 X\_year\_2010 = **None**;  
 y\_year\_2010 = **None  
 for** i **in** range(data\_2010.shape[1] - 3):  
 **if** X\_year\_2010 **is None**:  
 X\_year\_2010 = data\_2010[:, i:i + 3]  
 y\_year\_2010 = data\_2010[:, i + 3]  
 **else**:  
 X\_year\_2010 = np.concatenate((X\_year\_2010, data\_2010[:, i:i + 3]), axis=0)  
 y\_year\_2010 = np.concatenate((y\_year\_2010, data\_2010[:, i + 3]), axis=0)  
  
 *# test 2005* temp = data[[**'SUBDIVISION'**, **'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**,  
 **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].loc[data[**'YEAR'**] == 2005]  
  
 data\_2005 = np.asarray(temp[[**'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**,  
 **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].loc[temp[**'SUBDIVISION'**] == **'TELANGANA'**])  
  
 X\_year\_2005 = **None**;  
 y\_year\_2005 = **None  
 for** i **in** range(data\_2005.shape[1] - 3):  
 **if** X\_year\_2005 **is None**:  
 X\_year\_2005 = data\_2005[:, i:i + 3]  
 y\_year\_2005 = data\_2005[:, i + 3]  
 **else**:  
 X\_year\_2005 = np.concatenate((X\_year\_2005, data\_2005[:, i:i + 3]), axis=0)  
 y\_year\_2005 = np.concatenate((y\_year\_2005, data\_2005[:, i + 3]), axis=0)  
 *# terst 2015* temp = data[[**'SUBDIVISION'**, **'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**,  
 **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].loc[data[**'YEAR'**] == 2015]  
  
 data\_2015 = np.asarray(temp[[**'JAN'**, **'FEB'**, **'MAR'**, **'APR'**, **'MAY'**, **'JUN'**, **'JUL'**,  
 **'AUG'**, **'SEP'**, **'OCT'**, **'NOV'**, **'DEC'**]].loc[temp[**'SUBDIVISION'**] == **'TELANGANA'**])  
  
 X\_year\_2015 = **None**;  
 y\_year\_2015 = **None  
 for** i **in** range(data\_2015.shape[1] - 3):  
 **if** X\_year\_2015 **is None**:  
 X\_year\_2015 = data\_2015[:, i:i + 3]  
 y\_year\_2015 = data\_2015[:, i + 3]  
 **else**:  
 X\_year\_2015 = np.concatenate((X\_year\_2015, data\_2015[:, i:i + 3]), axis=0)  
 y\_year\_2015 = np.concatenate((y\_year\_2015, data\_2015[:, i + 3]), axis=0)  
  
 **from** sklearn **import** linear\_model  
  
 *# linear model* reg = linear\_model.ElasticNet(alpha=0.5)  
 reg.fit(X\_train, y\_train)  
 y\_pred = reg.predict(X\_test)  
 print(mean\_absolute\_error(y\_test, y\_pred))  
  
 y\_year\_pred\_2005 = reg.predict(X\_year\_2005)  
  
 *# 2010* y\_year\_pred\_2010 = reg.predict(X\_year\_2010)  
  
 y\_year\_pred\_2015 = reg.predict(X\_year\_2015)  
  
 print(**"MEAN 2005"**)  
 print(np.mean(y\_year\_2005), np.mean(y\_year\_pred\_2005))  
 print(**"Standard deviation 2005"**)  
 print(np.sqrt(np.var(y\_year\_2005)), np.sqrt(np.var(y\_year\_pred\_2005)))  
  
 print(**"MEAN 2010"**)  
 print(np.mean(y\_year\_2010), np.mean(y\_year\_pred\_2010))  
 print(**"Standard deviation 2010"**)  
 print(np.sqrt(np.var(y\_year\_2010)), np.sqrt(np.var(y\_year\_pred\_2010)))  
  
 print(**"MEAN 2015"**)  
 print(np.mean(y\_year\_2015), np.mean(y\_year\_pred\_2015))  
 print(**"Standard deviation 2015"**)  
 print(np.sqrt(np.var(y\_year\_2015)), np.sqrt(np.var(y\_year\_pred\_2015)))  
  
 plot\_graphs(y\_year\_2005, y\_year\_pred\_2005, **"Year-2005"**)  
 plot\_graphs(y\_year\_2010, y\_year\_pred\_2010, **"Year-2010"**)  
 plot\_graphs(y\_year\_2015, y\_year\_pred\_2015, **"Year-2015"**)  
 plt.close()  
  
  **def** testMltMSE(self,data):  
 rsltdict = {}  
 X = data[**'JAN'**].values  
 y = data[**'DEC'**].values  
 X = X.reshape(-1, 1)  
 y = y.reshape(-1, 1)  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)  
 **from** sklearn.linear\_model **import** LinearRegression  
 regressor = LinearRegression()  
 regressor.fit(X\_train, y\_train)  
 y\_pred = regressor.predict(X\_test)  
 *# print('Predection result ',y\_pred)  
 # print('Test ',X\_test)* mse = metrics.mean\_squared\_error(y\_test, y\_pred)  
 rmse = np.sqrt(mse)  
 lrcorr = regressor.coef\_  
 lcoe = sum(lrcorr) / len(lrcorr)  
 rsltdict.update({**'lgmse'**: mse, **'lgrmse'**: rmse, **'lgcorr'**: lcoe})  
 accuracy = metrics.accuracy\_score(y\_test.round(), y\_pred.round(), normalize=**False**)  
 *# print('LR accuracy ', accuracy)  
  
 # print(mse, "==", rmse)* 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=0.1, random\_state=42)  
 **from** sklearn.linear\_model **import** LinearRegression  
 regressor = LinearRegression()  
 regressor.fit(X\_train, y\_train)  
 y\_pred = regressor.predict(X\_test)  
 *# print('Predection result ',y\_pred)  
 # print('Test ',X\_test)* mse = metrics.mean\_squared\_error(y\_test, y\_pred)  
 rmse = np.sqrt(mse)  
 mlrco = regressor.coef\_  
 mlrcorre = sum(mlrco) / len(mlrco)  
 *# accuracy = metrics.accuracy\_score(y\_test.round(), y\_pred.round(),normalize=False)  
 # print('MLR accuracy ',accuracy)* print(mse, **"=="**, rmse)  
 *# plt.scatter(x=y\_test, y=y\_pred, c="red")  
 # plt.show()* rsltdict.update({**'mlrmse'**: mse, **'mlrrmse'**: rmse, **'mlrcorr'**: lcoe})  
  
 *####QPF COde* **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'**])  
 model.summary()  
  
 model.fit(x=np.expand\_dims(X\_train, axis=2), y=y\_train, batch\_size=64, epochs=10, verbose=1,  
 validation\_split=0.1,  
 shuffle=**True**)  
 y\_pred = model.predict(np.expand\_dims(X\_test, axis=2))  
 qmse = metrics.mean\_squared\_error(y\_test, y\_pred)  
 qrmse = np.sqrt(qmse)  
 print(**'Qmse = '**, qmse, **" QRmse="**, qrmse)  
 qcorr = qmse / qrmse  
 rsltdict.update({**'qpfmse'**: qmse, **'qpfrmse'**: qrmse, **'qpfcorr'**: qcorr})  
 *# accuracy = metrics.accuracy\_score(y\_test.round(), y\_pred.round(),normalize=False)  
 # print('QPF accuracy ',accuracy)* plt.close()  
 **return** rsltdict

**users side views.py**

**import** math  
  
**from** django.shortcuts **import** render,HttpResponse  
**from** sklearn.model\_selection **import** train\_test\_split  
  
**from** .forms **import** UserRegistrationForm  
**import** requests  
**from** django.contrib **import** messages  
**from** .forms **import** UserRegistrationForm  
**from** .models **import** RainFallUserRegistrationModel,IndiaRainFallDataModel  
**import** io,csv  
**from** django\_pandas.io **import** read\_frame  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.pyplot **as** plt  
**import** seaborn **as** sns; sns.set()  
**from** sklearn.preprocessing **import** StandardScaler  
**import** numpy **as** np *# linear algebra***import** pandas **as** pd *# data processing, CSV file I/O (e.g. pd.read\_csv)***from** sklearn **import** metrics  
**import** os  
**from** .GenGraphCode **import** GeneratePltGraph  
BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))  
*# Create your views here.***def** UserLogin(request):  
 **return** render(request,**'UserLogin.html'**,{})  
  
**def** UserRegister(request):  
 form = UserRegistrationForm()  
 **return** render(request,**'UserRegisterForm.html'**,{**'form'**:form})  
  
  
**def** GetWeatherInfo(request):  
 **if** request.method == **'POST'**:  
 city = request.POST.get(**'cityname'**)  
 accesKey = **'9c5781db8bb4ee7c96f7dee77728e353'** url = **'http://api.openweathermap.org/data/2.5/weather?q={}&units=imperial&appid=9c5781db8bb4ee7c96f7dee77728e353'** *#city = 'Las Vegas'* city\_weather = requests.get( url.format(city)).json() *# request the API data and convert the JSON to Python data types* print(**"Weather info "**,city\_weather)  
 weather = {  
 **'city'**: city,  
 **'temperature'**: city\_weather[**'main'**][**'temp'**],  
 **'description'**: city\_weather[**'weather'**][0][**'description'**],  
 **'icon'**: city\_weather[**'weather'**][0][**'icon'**],  
 *#'message':city\_weather['message']* }  
  
 context = {**'weather'**: weather}  
 **return** render(request, **'weatherinfo.html'**, context)  
  
**def** UserRegisterAction(request):  
 **if** request.method == **'POST'**:  
 form = UserRegistrationForm(request.POST)  
 **if** form.is\_valid():  
 print(**'Data is Valid'**)  
 form.save()  
 messages.success(request, **'You have been successfully registered'**)  
 *# return HttpResponseRedirect('./CustLogin')* form = UserRegistrationForm()  
 **return** render(request, **'UserRegisterForm.html'**, {**'form'**: form})  
 **else**:  
 print(**"Invalid form"**)  
 **else**:  
 form = UserRegistrationForm()  
 **return** render(request, **'UserRegisterForm.html'**, {**'form'**: form})  
  
  
**def** UserLoginCheck(request):  
 **if** request.method == **"POST"**:  
 loginid = request.POST.get(**'loginid'**)  
 pswd = request.POST.get(**'pswd'**)  
 print(**"Login ID = "**, loginid, **' Password = '**, pswd)  
 **try**:  
 check = RainFallUserRegistrationModel.objects.get(loginid=loginid, password=pswd)  
 status = check.status  
 print(**'Status is = '**, status)  
 **if** status == **"activated"**:  
 request.session[**'id'**] = check.id  
 request.session[**'loggeduser'**] = check.name  
 request.session[**'loginid'**] = loginid  
 request.session[**'email'**] = check.email  
 print(**"User id At"**, check.id, status)  
 **return** render(request, **'users/UserHomePage.html'**, {})  
 **else**:  
 messages.success(request, **'Your Account Not at activated'**)  
 **return** render(request, **'UserLogin.html'**)  
 *# return render(request, 'user/userpage.html',{})* **except** Exception **as** e:  
 print(**'Exception is '**, str(e))  
 **pass** messages.success(request, **'Invalid Login id and password'**)  
 **return** render(request, **'UserLogin.html'**, {})  
  
**def** GetWeatherInfo(request):  
 **if** request.method == **'POST'**:  
 city = request.POST.get(**'cityname'**)  
 accesKey = **'9c5781db8bb4ee7c96f7dee77728e353'** url = **'http://api.openweathermap.org/data/2.5/weather?q={}&units=imperial&appid=9c5781db8bb4ee7c96f7dee77728e353'** city\_weather = requests.get( url.format(city)).json() *# request the API data and convert the JSON to Python data types* print(**"Weather info "**,city\_weather)  
 code = city\_weather[**'cod'**]  
 **if** code==200:  
 print(**'Data valid'**)  
 **else**:  
 messages.success(request, **'Data Not Found'**)  
 **return** render(request, **'users/GetWeatherInfo.html'**, {})  
 print(**'Status code '**,code)  
 weather = {  
 **'city'**: city,  
 **'temperature'**: city\_weather[**'main'**][**'temp'**],  
 **'description'**: city\_weather[**'weather'**][0][**'description'**],  
 **'icon'**: city\_weather[**'weather'**][0][**'icon'**],  
 *#'message':city\_weather['message']* }  
  
 context = {**'weather'**: weather}  
 **return** render(request, **'users/GetWeatherInfo.html'**, context)  
  
**def** SearchByCity(request):  
 **return** render(request,**'users/SearchByCity.html'**,{})  
  
**def** UserUploadData(request):  
 **return** render(request,**'users/uploaddata.html'**,{})  
  
  
**def** UploadCSVToDataBase(request):  
 *# declaring template* template = **"users/UserHomePage.html"** data = IndiaRainFallDataModel.objects.all()  
 *# prompt is a context variable that can have different values depending on their context* prompt = {  
 **'order'**: **'Order of the CSV should be name, email, address, phone, profile'**,  
 **'profiles'**: data  
 }  
 *# GET request returns the value of the data with the specified key.* **if** request.method == **"GET"**:  
 **return** render(request, template, prompt)  
 csv\_file = request.FILES[**'file'**]  
 *# let's check if it is a csv file* **if not** csv\_file.name.endswith(**'.csv'**):  
 messages.error(request, **'THIS IS NOT A CSV FILE'**)  
 data\_set = csv\_file.read().decode(**'UTF-8'**)  
  
 *# setup a stream which is when we loop through each line we are able to handle a data in a stream* io\_string = io.StringIO(data\_set)  
 next(io\_string)  
 **for** column **in** csv.reader(io\_string, delimiter=**','**, quotechar=**"|"**):  
 \_, created = IndiaRainFallDataModel.objects.update\_or\_create(  
 SUBDIVISION=column[0],  
 YEAR=column[1],  
 JAN=column[2],  
 FEB=column[3],  
 MAR=column[4],  
 APR=column[5],  
 MAY=column[6],  
 JUN=column[7],  
 JUL=column[8],  
 AUG=column[9],  
 SEP=column[10],  
 OCT=column[11],  
 NOV=column[12],  
 DEC=column[13],  
 ANNUAL=column[14],  
 JanToFeb=column[15],  
 MarToMay=column[16],  
 JunToSep=column[17],  
 OctToDec=column[18]  
  
 )  
 context = {}  
  
 **return** render(request, **'users/UserHomePage.html'**, context)  
  
**def** UserDataPreProcess(request):  
 qs = IndiaRainFallDataModel.objects.all()  
 data = read\_frame(qs)  
 g = GeneratePltGraph()  
 g.preProcessGraphs(data)  
 **return** render(request,**'users/PreProcessedData.html'**,{**'data'**:qs})  
  
**def** UserMLRCode(request):  
 qs = IndiaRainFallDataModel.objects.all()  
 data = read\_frame(qs)  
 gf = GeneratePltGraph()  
 gf.genMlrCodes(data)  
 **return** render(request,**'users/UsersMachineLearningGraphs.html'**,{})  
  
  
  
  
**def** TestMlR(request):  
  
 qs = IndiaRainFallDataModel.objects.all()  
 data = read\_frame(qs)  
 gh = GeneratePltGraph()  
 rsltdict = gh.testMltMSE(data)  
 **return** render(request,**"users/MlrTestResult.html"**,rsltdict)

user side **models.py**

**from** django.db **import** models  
  
*# Create your models here.***class** RainFallUserRegistrationModel(models.Model):  
 name = models.CharField(max\_length=100)  
 loginid = models.CharField(unique=**True**,max\_length=100)  
 password = models.CharField(max\_length=100)  
 mobile = models.CharField(max\_length=100)  
 email = models.CharField(max\_length=100)  
 locality = models.CharField(max\_length=100)  
 address = models.CharField(max\_length=1000)  
 city = models.CharField(max\_length=100)  
 state = models.CharField(max\_length=100)  
 status = models.CharField(max\_length=100)  
  
 **def** \_\_str\_\_(self):  
 **return** self.loginid  
 **class** Meta:  
 db\_table=**'RainFallUsers'  
  
  
class** IndiaRainFallDataModel(models.Model):  
 SUBDIVISION = models.CharField(max\_length=100)  
 YEAR = models.IntegerField()  
 JAN = models.FloatField(default=0.0);  
 FEB = models.FloatField(default=0.0);  
 MAR = models.FloatField(default=0.0);  
 APR = models.FloatField(default=0.0);  
 MAY = models.FloatField(default=0.0);  
 JUN = models.FloatField(default=0.0);  
 JUL = models.FloatField(default=0.0);  
 AUG = models.FloatField(default=0.0);  
 SEP = models.FloatField(default=0.0);  
 OCT = models.FloatField(default=0.0);  
 NOV = models.FloatField(default=0.0);  
 DEC = models.FloatField(default=0.0);  
 ANNUAL = models.FloatField(default=0.0);  
 JanToFeb = models.FloatField(default=0.0);  
 MarToMay = models.FloatField(default=0.0);  
 JunToSep = models.FloatField(default=0.0);  
 OctToDec = models.FloatField(default=0.0);  
 **def** \_\_str\_\_(self):  
 **return** self.SUBDIVISION  
  
 **class** Meta:  
 db\_table = **'IndianRainfall'**

user side **forms.py**

**from** django **import** forms  
**from** .models **import** RainFallUserRegistrationModel  
  
**class** UserRegistrationForm(forms.ModelForm):  
 name = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**:**'[a-zA-Z]+'**}), required=**True**,max\_length=100)  
 loginid = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**:**'[a-zA-Z]+'**}), required=**True**,max\_length=100)  
 password = forms.CharField(widget=forms.PasswordInput(attrs={**'pattern'**:**'(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z]).{8,}'**,**'title'**:**'Must contain at least one number and one uppercase and lowercase letter, and at least 8 or more characters'**}), required=**True**,max\_length=100)  
 mobile = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**:**'[56789][0-9]{9}'**}), required=**True**,max\_length=100)  
 email = forms.CharField(widget=forms.TextInput(attrs={**'pattern'**:**'[a-z0-9.\_%+-]+@[a-z0-9.-]+\.[a-z]{2,}$'**}), required=**True**,max\_length=100)  
 locality = forms.CharField(widget=forms.TextInput(), required=**True**,max\_length=100)  
 address = forms.CharField(widget=forms.Textarea(attrs={**'rows'**:4, **'cols'**: 22}), required=**True**,max\_length=250)  
 city = forms.CharField(widget=forms.TextInput(attrs={**'class'**:**'form-control'** , **'autocomplete'**: **'off'**,**'pattern'**:**'[A-Za-z ]+'**, **'title'**:**'Enter Characters Only '**}), required=**True**,max\_length=100)  
 state = forms.CharField(widget=forms.TextInput(attrs={**'class'**:**'form-control'** , **'autocomplete'**: **'off'**,**'pattern'**:**'[A-Za-z ]+'**, **'title'**:**'Enter Characters Only '**}), required=**True**,max\_length=100)  
 status = forms.CharField(widget=forms.HiddenInput(), initial=**'waiting'** ,max\_length=100)  
  
  
 **class** Meta():  
 model = RainFallUserRegistrationModel  
 fields=**'\_\_all\_\_'**

admin side **views.py**

**from** django.shortcuts **import** render,HttpResponse  
**from** django.contrib **import** messages  
*# Create your views here.***from** users.models **import** RainFallUserRegistrationModel,IndiaRainFallDataModel  
**from** .forms **import** IndiaRainFallDataForm  
**from** django.core.paginator **import** Paginator, EmptyPage, PageNotAnInteger  
  
**def** AdminLogin(request):  
 **return** render(request,**'AdminLogin.html'**,{})  
  
**def** AdminLoginCheck(request):  
 **if** request.method == **'POST'**:  
 usrid = request.POST.get(**'loginname'**)  
 pswd = request.POST.get(**'pswd'**)  
 print(**"User ID is = "**, usrid)  
 **if** usrid == **'admin' and** pswd == **'admin'**:  
 **return** render(request, **'admins/AdminHome.html'**)  
  
 **else**:  
 messages.success(request, **'Please Check Your Login Details'**)  
 **return** render(request, **'AdminLogin.html'**, {})  
  
  
**def** AdminViewUsers(request):  
 data = RainFallUserRegistrationModel.objects.all()  
 **return** render(request,**'admins/AdminViewUsers.html'**,{**'data'**:data})  
  
**def** AdminActivaUsers(request):  
 **if** request.method == **'GET'**:  
 id = request.GET.get(**'uid'**)  
 status = **'activated'** print(**"PID = "**, id, status)  
 RainFallUserRegistrationModel.objects.filter(id=id).update(status=status)  
 data = RainFallUserRegistrationModel.objects.all()  
 **return** render(request,**'admins/AdminViewUsers.html'**,{**'data'**:data})  
  
**def** AdminAddData(request):  
 **if** request.method == **'POST'**:  
 form = IndiaRainFallDataForm(request.POST)  
 **if** form.is\_valid():  
 print(**'Data is Valid'**)  
 form.save()  
 messages.success(request, **'Data Added Successfull'**)  
 form = IndiaRainFallDataForm()  
 **return** render(request, **'admins/AddRainFallData.html'**, {**'form'**: form})  
 **else**:  
 print(**"Invalid form"**)  
 **else**:  
 form = IndiaRainFallDataForm()  
 **return** render(request, **'admins/AddRainFallData.html'**, {**'form'**: form})  
  
**def** AdminViewData(request):  
 data\_list = IndiaRainFallDataModel.objects.all()  
 page = request.GET.get(**'page'**, 1)  
  
 paginator = Paginator(data\_list, 60)  
 **try**:  
 users = paginator.page(page)  
 **except** PageNotAnInteger:  
 users = paginator.page(1)  
 **except** EmptyPage:  
 users = paginator.page(paginator.num\_pages)  
  
 **return** render(request, **'admins/AdminViewWeather.html'**, {**'users'**: users})  
  
  
**base.html**

{%load static%}  
<!DOCTYPE **html**>  
  
<**html lang="en"**>  
 <**head**>  
 <**meta charset="UTF-8"**>  
 <**meta http-equiv="X-UA-Compatible" content="IE=edge"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1.0,maximum-scale=1"**>  
 <**meta name="viewport" content="width=device-width, initial-scale=1"**>  
 <**link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"**>  
 <**script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"**></**script**>  
 <**script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.16.0/umd/popper.min.js"**></**script**>  
 <**script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.5.0/js/bootstrap.min.js"**></**script**>  
  
 <**title**>Compass Starter by India, Rian</**title**>  
  
 <**link href="http://fonts.googleapis.com/css?family=Roboto:300,400,700|" rel="stylesheet" type="text/css"**>  
 <**link href="{%static 'fonts/font-awesome.min.css'%}" rel="stylesheet" type="text/css"**>  
  
 <**link rel="stylesheet" href="{%static 'style.css'%}"**>  
  
 </**head**>  
  
  
 <**body**>  
  
 <**div class="site-content"**>  
 <**div class="site-header"**>  
 <**div class="container"**>  
 <**a href="index.html" class="branding"**>  
 <**img src="{%static 'images/logo.png'%}" alt="" class="logo"**>  
 <**div class="logo-type"**>  
 <**h1 class="site-title"**>Rainfall Prediction</**h1**>  
 <**small class="site-description"**>Machine Learning based</**small**>  
 </**div**>  
 </**a**>  
  
 *<!-- Default snippet for navigation -->* <**div class="main-navigation"**>  
 <**button type="button" class="menu-toggle"**><**i class="fa fa-bars"**></**i**></**button**>  
 <**ul class="menu"**>  
 <**li class="menu-item"**><**a href="{%url 'index'%}"**>Home</**a**></**li**>  
 <**li class="menu-item"**><**a href="{%url 'UserLogin'%}"**>Users</**a**></**li**>  
 <**li class="menu-item"**><**a href="{%url 'AdminLogin'%}"**>Admins</**a**></**li**>  
 <**li class="menu-item"**><**a href="{%url 'UserRegister'%}"**>Register</**a**></**li**>  
  
 </**ul**>  
 </**div**>  
  
 <**div class="mobile-navigation"**></**div**>  
  
 </**div**>  
 </**div**> *<!-- .site-header -->* <**div class="hero" data-bg-image="{%static 'images/banner1.png'%}"**>  
 <**div class="container"**>  
 {%block contents%}  
  
 {%endblock%}  
  
  
 </**div**>  
 </**div**>  
  
  
 <**footer class="site-footer"**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col-md-8"**>  
  
 </**div**>  
 <**div class="col-md-3 col-md-offset-1"**>  
  
 </**div**>  
 </**div**>  
  
 <**p class="colophon"**>Copyright 2020 Alex Corporation. Designed by Alex. All rights reserved</**p**>  
 </**div**>  
 </**footer**> *<!-- .site-footer -->* </**div**>  
  
 <**script src="{%static 'js/jquery-1.11.1.min.js'%}"**></**script**>  
 <**script src="{%static 'js/plugins.js'%}"**></**script**>  
 <**script src="{%static 'js/app.js'%}"**></**script**>  
  
 </**body**>  
  
</**html**>

**userregisterform.html**

{%extends 'base.html'%}  
{%block contents %}  
<**center**>  
 <**p**>  
 <**form method="POST" action="{%url 'UserRegisterAction'%}"** >  
 {% csrf\_token %}  
<**h2**>User Register Form</**h2**>  
 <**table**>  
 <**tr**><**td**>Customer Name</**td**><**td**>{{form.name}}</**td**></**tr**>  
 <**tr**><**td**>Login ID</**td**><**td**>{{form.loginid}}</**td**></**tr**>  
 <**tr**><**td**>Password</**td**><**td**>{{form.password}}</**td**></**tr**>  
 <**tr**><**td**>Mobile</**td**><**td**>{{form.mobile}}</**td**></**tr**>  
 <**tr**><**td**>email</**td**><**td**>{{form.email}}</**td**></**tr**>  
 <**tr**><**td**>Locality</**td**><**td**>{{form.locality}}</**td**></**tr**>  
 <**tr**><**td**>Address</**td**><**td**>{{form.address}}</**td**></**tr**>  
 <**tr**><**td**>City</**td**><**td**>{{form.city}}</**td**></**tr**>  
 <**tr**><**td**>State</**td**><**td**>{{form.state}}</**td**></**tr**>  
 <**tr**><**td**></**td**><**td**>{{form.status}}</**td**></**tr**>  
  
 <**tr**><**td**><**button class="btn btn-primary my-2 my-sm-0" type="submit" style = "**margin-left:20%;**"**>Register</**button**></**td**></**tr**>  
  
 {% if messages %}  
 {% for message in messages %}  
 <**font color='GREEN'**> {{ message }}</**font**>  
 {% endfor %}  
 {% endif %}  
  
 </**table**>  
  
 </**form**>  
  
 </**p**>  
 </**center**>  
 {%endblock%}

**Urls.py**

*"""RainfallPredection URL Configuration  
  
The `urlpatterns` list routes URLs to views. For more information please see:  
 https://docs.djangoproject.com/en/2.0/topics/http/urls/  
Examples:  
Function views  
 1. Add an import: from my\_app import views  
 2. Add a URL to urlpatterns: path('', views.home, name='home')  
Class-based views  
 1. Add an import: from other\_app.views import Home  
 2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')  
Including another URLconf  
 1. Import the include() function: from django.urls import include, path  
 2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))  
"""***from** django.contrib **import** admin  
**from** django.urls **import** path  
**from** RainfallPredection **import** views **as** mainView  
**from** users **import** views **as** usr  
**from** admins **import** views **as** admns  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 path(**''**, mainView.index, name=**'index'**),  
 path(**'Logout/'**,mainView.Logout,name=**'Logout'**),  
  
 *### User Side all urs* path(**'UserLogin/'**,usr.UserLogin,name=**'UserLogin'**),  
 path(**'UserRegister/'**,usr.UserRegister,name=**'UserRegister'**),  
 path(**'GetWeatherInfo/'**,usr.GetWeatherInfo,name=**'GetWeatherInfo'**),  
 path(**'UserRegisterAction/'**,usr.UserRegisterAction,name=**'UserRegisterAction'**),  
 path(**'UserLoginCheck/'**,usr.UserLoginCheck, name=**'UserLoginCheck'**),  
 path(**'GetWeatherInfo/'**,usr.GetWeatherInfo, name=**'GetWeatherInfo'**),  
 path(**'SearchByCity/'**,usr.SearchByCity, name=**'SearchByCity'**),  
 path(**'UserUploadData/'**,usr.UserUploadData,name=**'UserUploadData'**),  
 path(**'UploadCSVToDataBase/'**, usr.UploadCSVToDataBase, name=**'UploadCSVToDataBase'**),  
 path(**'UserDataPreProcess/'**,usr.UserDataPreProcess, name=**'UserDataPreProcess'**),  
 path(**'UserMLRCode/'**,usr.UserMLRCode,name=**'UserMLRCode'**),  
 path(**'TestMlR/'**,usr.TestMlR, name=**'TestMlR'**),  
  
  
 *### Admins side urls* path(**'AdminLogin/'**,admns.AdminLogin,name=**'AdminLogin'**),  
 path(**'AdminLoginCheck/'**,admns.AdminLoginCheck,name=**'AdminLoginCheck'**),  
 path(**'AdminViewUsers/'**, admns.AdminViewUsers, name=**'AdminViewUsers'**),  
 path(**'AdminActivaUsers/'**,admns.AdminActivaUsers,name=**'AdminActivaUsers'**),  
 path(**'AdminAddData/'**,admns.AdminAddData,name=**'AdminAddData'**),  
 path(**'AdminViewData/'**,admns.AdminViewData,name=**'AdminViewData'**),  
  
]