**SOURCE CODE**

User Side views.py

**from** django.shortcuts **import** render,HttpResponse,redirect  
**from** django.contrib **import** messages  
**from** .forms **import** UserRegistrationForm  
**from** .models **import** UserRegistrationModel,AutoMLDataModel,MyPredectionsModels,ModelPredectionStoreModels  
**from** .UserAutoMachineLearningProcess **import** StartProcessAutoML  
**import** h2o  
**from** django.core.paginator **import** Paginator, EmptyPage, PageNotAnInteger  
**import** csv,io  
**from** django\_pandas.io **import** read\_frame  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
*# Create your views here.***def** UserRegisterActions(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'**)  
 form = UserRegistrationForm()  
 **return** render(request, **'UsersRegister.html'**, {**'form'**: form})  
 **else**:  
 messages.success(request, **'Email or Mobile Already Existed'**)  
 print(**"Invalid form"**)  
 **else**:  
 form = UserRegistrationForm()  
 **return** render(request, **'UsersRegister.html'**, {**'form'**: form})  
  
  
**def** UserLoginCheck(request):  
 **if** request.method == **"POST"**:  
 loginid = request.POST.get(**'loginname'**)  
 pswd = request.POST.get(**'pswd'**)  
 print(**"Login ID = "**, loginid, **' Password = '**, pswd)  
 **try**:  
 check = UserRegistrationModel.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/UserHome.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** UserHome(request):  
 **return** render(request, **'users/UserHome.html'**, {})  
  
**def** UserAutoMLTest(request):  
 obj = StartProcessAutoML()  
 html = **''** data1=**''  
 try**:  
  
 **pass** lb = obj.startDataPreprocess()  
 data\_as\_df = h2o.as\_list(lb)  
 html = data\_as\_df.to\_html()  
 *#data1 = data.to\_html()* **except** Exception **as** e:  
 **pass** data\_list = AutoMLDataModel.objects.all()  
 *#print("Lb type is ",type(lb))* **return** render(request,**"users/AutoMachineLearning.html"**,{**"html"**:html,**"dataset"**:data\_list})  
 *#return HttpResponse("Exit code 0")  
 #return redirect('AutoResponse')***def** AutoResponse(request):  
 data\_list = AutoMLDataModel.objects.all()  
 page = request.GET.get(**'page'**, 1)  
  
 paginator = Paginator(data\_list, 10)  
 **try**:  
 users = paginator.page(page)  
 **except** PageNotAnInteger:  
 users = paginator.page(1)  
 **except** EmptyPage:  
 users = paginator.page(paginator.num\_pages)  
 **return** render(request, **'users/AutoMachineLearning.html'**, {**'users'**: users})  
  
**def** DataUploadForm(request):  
 **return** render(request,**'users/useruploaddata.html'**,{})  
  
**def** UploadDatatoServer(request):  
 AutoMLDataModel  
 *# declaring template* template = **"users/useruploaddata.html"** data = AutoMLDataModel.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=**'\t'**, quotechar=**"|"**):  
 print(**"Data is = "**,column[0])  
 \_, created = AutoMLDataModel.objects.update\_or\_create(  
 Age=column[1],  
 Workclass=column[2],  
 EducationNum=column[3],  
 MaritalStatus=column[4],  
 Occupation=column[5],  
 Relationship=column[6],  
 Race=column[7],  
 Sex=column[8],  
 CapitalGain=column[9],  
 CapitalLoss=column[10],  
 Hoursperweek=column[11],  
 Country=column[12]  
  
  
 )  
 context = {}  
  
 **return** render(request, **'users/useruploaddata.html'**, context)  
  
**def** UploadDatatoServerForPredections(request):  
 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=**"|"**):  
 print(**"Data is = "**, column[0])  
 \_, created = MyPredectionsModels.objects.update\_or\_create(  
 YearsExperience=column[0],  
 Salary=column[1]  
  
 )  
 context = {}  
  
 **return** render(request, **'users/useruploaddata.html'**, context)  
  
**def** MyPredectionsSlot1(request):  
 data = MyPredectionsModels.objects.all()  
 **return** render(request,**'users/MyPredections.html'**,{**'data'**:data})  
  
**def** MyPredectionsSlot2(request):  
 data = MyPredectionsModels.objects.all()  
 **return** render(request, **'users/DataSlot1.html'**, {**'data'**: data})  
  
  
**def** MyPredectionsSlot3(request):  
 **if** request.method==**'POST'**:  
 splitsize = int(request.POST.get(**'testsize'**))  
 testsize = splitsize/100  
 data = MyPredectionsModels.objects.all()  
 dataset = read\_frame(data)  
 X = dataset.iloc[:, :1].values  
 y = dataset.iloc[:, -1].values  
 **from** sklearn.model\_selection **import** train\_test\_split  
 X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=testsize, random\_state=0)  
 *#print('X\_train', X\_train)  
 #print('X\_test', X\_test)  
 #print('y\_train', y\_train)* **from** sklearn.linear\_model **import** LinearRegression  
 model = LinearRegression()  
 model.fit(X\_train, y\_train)  
  
 plt.scatter(X\_test, y\_test, color=**'red'**)  
 plt.plot(X\_train, model.predict(X\_train), color=**'blue'**)  
 plt.title(**'Salary vs Experience (Test set)'**)  
 plt.xlabel(**'Years of Experience'**)  
 plt.ylabel(**'Salary'**)  
 plt.show()  
 score = model.score(X, y)  
 loginid = request.session[**'loginid'**]  
 email = request.session[**'email'**]  
 ModelPredectionStoreModels.objects.create(username=loginid, email=email, acheiveaccuracy=score,testsize=testsize)  
 y\_pred = model.predict(X\_test)  
 y\_pred = np.around(y\_pred, 1)  
 print(**'predected Result '**, type(y\_pred.tolist()))  
 print(**'Original salary '**, type(y\_test))  
 myDict = {**'original'**:y\_test.tolist(),**'predections'**:y\_pred.tolist()}  
 print(**"My Dict "**,myDict)  
 **return** render(request,**'users/DataSlot2.html'**,{**'data'**:myDict})  
 *#return HttpResponse(html)***def** AddDataForm(request):  
 data = MyPredectionsModels.objects.all()  
 **return** render(request, **'users/AddDataForm.html'**, {**'data'**: data})  
  
  
**def** AddDataToDataset(request):  
 **if** request.method==**'POST'**:  
 exp = request.POST.get(**'Experience'**)  
 salary = request.POST.get(**'salary'**)  
 MyPredectionsModels.objects.create(YearsExperience=exp,Salary=salary)  
 data = MyPredectionsModels.objects.all()  
 **return** render(request, **'users/DataSlot1.html'**, {**'data'**: data})

**UserAutoMachineLearningProcess.py**

**import** sklearn  
**import** pandas **as** pd  
**import** numpy **as** np  
**import** shap  
**import** h2o  
**from** h2o.automl **import** H2OAutoML  
*#h2o.init(nthreads=-1)***class** H2OProbWrapper:  
 **def** \_\_init\_\_(self, h2o\_model, feature\_names):  
 self.h2o\_model = h2o\_model  
 self.feature\_names = feature\_names  
  
 **def** predict\_binary\_prob(self, X):  
 **if** isinstance(X, pd.Series):  
 X = X.values.reshape(1,-1)  
 self.dataframe= pd.DataFrame(X, columns=self.feature\_names)  
 self.predictions = self.h2o\_model.predict(h2o.H2OFrame(self.dataframe)).as\_data\_frame().values  
 **return** self.predictions.astype(**'float64'**)[:,-1] *#probability of True class***class** StartProcessAutoML:  
 **def** startDataPreprocess(self):  
 h2o.init(nthreads=-1) *### Start the h20 Server* X, y = shap.datasets.adult()  
 X\_display, y\_display = shap.datasets.adult(display=**True**)  
 *#X\_display.to\_csv("output.csv", index=False)  
 #print(y\_display.shape)* print(X.head())  
 *#print(X\_display.dtypes)* X\_train, X\_test, y\_train, y\_test = sklearn.model\_selection.train\_test\_split(\*shap.datasets.adult(),  
 test\_size=0.2, random\_state=7)  
  
 train\_indices = X\_train.index  
 test\_indices = X\_test.index  
  
 X\_train\_display = X\_display.iloc[train\_indices]  
 y\_train\_display = y\_display[train\_indices]  
 X\_test\_display = X\_display.iloc[test\_indices]  
 y\_test\_display = y\_display[test\_indices]  
  
 X\_train.reset\_index(drop=**True**, inplace=**True**)  
 X\_test.reset\_index(drop=**True**, inplace=**True**)  
 X\_train\_display.reset\_index(drop=**True**, inplace=**True**)  
 X\_test\_display.reset\_index(drop=**True**, inplace=**True**)  
  
 train\_h2o\_df = h2o.H2OFrame(X\_train)  
 train\_h2o\_df[**'labels'**] = h2o.H2OFrame(y\_train)  
 train\_h2o\_df[**'labels'**] = train\_h2o\_df[**'labels'**].asfactor()  
 train\_h2o\_df[**'Workclass'**] = train\_h2o\_df[**'Workclass'**].asfactor()  
 train\_h2o\_df[**'Marital Status'**] = train\_h2o\_df[**'Marital Status'**].asfactor()  
 train\_h2o\_df[**'Relationship'**] = train\_h2o\_df[**'Relationship'**].asfactor()  
 train\_h2o\_df[**'Occupation'**] = train\_h2o\_df[**'Occupation'**].asfactor()  
 train\_h2o\_df[**'Sex'**] = train\_h2o\_df[**'Sex'**].asfactor()  
 train\_h2o\_df[**'Race'**] = train\_h2o\_df[**'Race'**].asfactor()  
 train\_h2o\_df[**'Country'**] = train\_h2o\_df[**'Country'**].asfactor()  
  
 test\_h2o\_df = h2o.H2OFrame(X\_test)  
 test\_h2o\_df[**'labels'**] = h2o.H2OFrame(y\_test)  
 test\_h2o\_df[**'labels'**] = test\_h2o\_df[**'labels'**].asfactor()  
 test\_h2o\_df[**'Workclass'**] = test\_h2o\_df[**'Workclass'**].asfactor()  
 test\_h2o\_df[**'Marital Status'**] = test\_h2o\_df[**'Marital Status'**].asfactor()  
 test\_h2o\_df[**'Relationship'**] = test\_h2o\_df[**'Relationship'**].asfactor()  
 test\_h2o\_df[**'Occupation'**] = test\_h2o\_df[**'Occupation'**].asfactor()  
 test\_h2o\_df[**'Sex'**] = test\_h2o\_df[**'Sex'**].asfactor()  
 test\_h2o\_df[**'Race'**] = test\_h2o\_df[**'Race'**].asfactor()  
 test\_h2o\_df[**'Country'**] = test\_h2o\_df[**'Country'**].asfactor()  
  
 feature\_names = list(X\_train.columns)  
  
 aml = H2OAutoML(max\_runtime\_secs=50, seed=2)  
 *#aml = H2OAutoML(max\_runtime\_secs=500, seed=42)* aml.train(x=feature\_names, y=**'labels'**, training\_frame=train\_h2o\_df)  
  
 lb = aml.leaderboard  
  
 print(lb)  
 bst\_model = aml.leader  
  
 h2o\_wrapper = H2OProbWrapper(bst\_model, feature\_names)  
  
 X\_train.shape[0]  
  
 explainer = shap.KernelExplainer(h2o\_wrapper.predict\_binary\_prob, X\_train.iloc[:100, :])  
  
 person = 0 *# first person in test dataset* print(**'prediction (probability that this person earns more than $50k/year) ='**,  
 h2o\_wrapper.predict\_binary\_prob(X\_test.iloc[person])[0])  
 print(**'ground\_truth (this person earns more than $50k/year) ='**, y\_test\_display[person])  
  
 shap.initjs()  
 shap\_values = explainer.shap\_values(X\_test.iloc[person, :], nsamples=500)  
 shap.force\_plot(explainer.expected\_value, shap\_values, X\_test\_display.iloc[person, :])  
  
 person = 1 *# second person in test dataset* print(**'prediction (probability that this person earns more than $50k/year) ='**,  
 h2o\_wrapper.predict\_binary\_prob(X\_test.iloc[person])[0])  
 print(**'ground\_truth (this person earns more than $50k/year) ='**, y\_test\_display[person])  
  
   
 shap.initjs()  
 shap\_values = explainer.shap\_values(X\_test.iloc[person, :], nsamples=500)  
 shap.force\_plot(explainer.expected\_value, shap\_values, X\_test\_display.iloc[person, :])  
 **'''  
 h2o.save\_model(bst\_model)  
 X\_test.to\_pickle('X\_test.pkl')  
 X\_train.to\_pickle('X\_train.pkl')  
 np.save('y\_test.npy', y\_test)  
 np.save('y\_train.npy', y\_train)  
 X\_test\_display.to\_pickle('X\_test\_display.pkl')  
 X\_train\_display.to\_pickle('X\_train\_display.pkl')  
 np.save('y\_test\_display.npy', y\_test\_display)  
 np.save('y\_train\_display.npy', y\_train\_display)  
 X.to\_pickle('X.pkl')'''** *#h2o.cluster().shutdown()* **return** lb

**models.py**

**from** django.db **import** models  
  
*# Create your models here.  
  
  
# Create your models here.***class** UserRegistrationModel(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(unique=**True**, max\_length=100)  
 email = models.CharField(unique=**True**, 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 = **'AutoUsers'  
  
  
  
class** AutoMLDataModel(models.Model):  
 id =models.IntegerField(primary\_key=**True**)  
 Age = models.FloatField(default=0)  
 Workclass = models.CharField(max\_length=200)  
 EducationNum = models.FloatField(default=0)  
 MaritalStatus = models.CharField(max\_length=200)  
 Occupation = models.CharField(max\_length=200)  
 Relationship = models.CharField(max\_length=200)  
 Race = models.CharField(max\_length=200)  
 Sex = models.CharField(max\_length=200)  
 CapitalGain= models.FloatField(default=0)  
 CapitalLoss = models.FloatField(default=0)  
 Hoursperweek = models.FloatField(default=0)  
 Country= models.CharField(max\_length=200)  
  
 *#def \_\_str\_\_(self):  
 #return self.id* **class** Meta:  
 db\_table = **'automldata'  
  
  
class** MyPredectionsModels(models.Model):  
 id = models.IntegerField(primary\_key=**True**)  
 YearsExperience = models.FloatField(default=0)  
 Salary = models.FloatField(default=0)  
  
 **def** \_\_str\_\_(self):  
 **return** self.id  
 **class** Meta:  
 db\_table = **'mypredections'  
  
class** ModelPredectionStoreModels(models.Model):  
 id = models.IntegerField(primary\_key=**True**)  
 username = models.CharField(max\_length=150)  
 email = models.CharField(max\_length=150)  
 acheiveaccuracy = models.FloatField()  
 testsize = models.FloatField()  
 cdata = models.DateTimeField(auto\_now\_add=**True**)

**forms.py**

**from** django **import** forms  
**from** .models **import** UserRegistrationModel  
  
**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={**'autocomplete'**: **'off'**,**'pattern'**:**'[A-Za-z ]+'**, **'title'**:**'Enter Characters Only '**}), required=**True**,max\_length=100)  
 state = forms.CharField(widget=forms.TextInput(attrs={**'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 = UserRegistrationModel  
 fields=**'\_\_all\_\_'**

**urls.py**

*"""AutomatedMachineLeanring 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** AutomatedMachineLeanring **import** views **as** mainView  
**from** users **import** views **as** usr  
**from** admins **import** views **as** admins  
  
urlpatterns = [  
 path(**'admin/'**, admin.site.urls),  
 path(**''**,mainView.index, name=**'index'**),  
 path(**'logout/'**,mainView.logout, name=**'logout'**),  
 path(**'UserLogin/'**,mainView.UserLogin, name=**'UserLogin'**),  
 path(**'AdminLogin/'**,mainView.AdminLogin, name=**'AdminLogin'**),  
 path(**'UserRegister/'**,mainView.UserRegister, name=**'UserRegister'**),  
  
 *### User Side Views ####* path(**'UserRegisterActions/'**, usr.UserRegisterActions, name=**'UserRegisterActions'**),  
 path(**'UserLoginCheck/'**, usr.UserLoginCheck, name=**'UserLoginCheck'**),  
 path(**'UserHome/'**, usr.UserHome, name=**'UserHome'**),  
 path(**'UserAutoMLTest/'**, usr.UserAutoMLTest, name=**'UserAutoMLTest'**),  
 path(**'UploadDatatoServer/'**,usr.UploadDatatoServer, name=**'UploadDatatoServer'**),  
 path(**'DataUploadForm/'**, usr.DataUploadForm, name=**'DataUploadForm'**),  
 path(**'AutoResponse/'**, usr.AutoResponse, name=**'AutoResponse'**),  
 path(**'UploadDatatoServerForPredections/'**, usr.UploadDatatoServerForPredections, name=**'UploadDatatoServerForPredections'**),  
 path(**'MyPredectionsSlot1/'**, usr.MyPredectionsSlot1, name=**'MyPredectionsSlot1'**),  
 path(**'MyPredectionsSlot2/'**, usr.MyPredectionsSlot2, name=**'MyPredectionsSlot2'**),  
 path(**'MyPredectionsSlot3/'**, usr.MyPredectionsSlot3, name=**'MyPredectionsSlot3'**),  
 path(**'AddDataToDataset/'**, usr.AddDataToDataset, name=**'AddDataToDataset'**),  
 path(**'AddDataForm/'**, usr.AddDataForm, name=**'AddDataForm'**),  
  
 *### Admin Side Urls ####* path(**'AdminLoginCheck/'**, admins.AdminLoginCheck, name=**'AdminLoginCheck'**),  
 path(**'AdminHome/'**, admins.AdminHome, name=**'AdminHome'**),  
 path(**'ViewUsersList/'**, admins.ViewUsersList, name=**'ViewUsersList'**),  
 path(**'AdminActivaUsers/'**, admins.AdminActivaUsers, name=**'AdminActivaUsers'**),  
 path(**'UserPerformedOperations/'**, admins.UserPerformedOperations, name=**'UserPerformedOperations'**),  
  
  
  
  
]

Adminside **views.py**

**from** django.shortcuts **import** render,HttpResponse  
**from** django.contrib **import** messages  
**from** users.forms **import** UserRegistrationModel  
**from** users.models **import** ModelPredectionStoreModels  
*# Create your views here.***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** AdminHome(request):  
 **return** render(request,**'admins/AdminHome.html'**,{})  
  
**def** ViewUsersList(request):  
 data = UserRegistrationModel.objects.all()  
 **return** render(request, **'admins/RegisteredUsers.html'**,{**'data'**:data})  
  
**def** AdminActivaUsers(request):  
 **if** request.method == **'GET'**:  
 id = request.GET.get(**'uid'**)  
 status = **'activated'** print(**"PID = "**, id, status)  
 UserRegistrationModel.objects.filter(id=id).update(status=status)  
 data = UserRegistrationModel.objects.all()  
 **return** render(request,**'admins/RegisteredUsers.html'**,{**'data'**:data})  
  
**def** UserPerformedOperations(request):  
 data = ModelPredectionStoreModels.objects.all()  
 **return** render(request, **'admins/UsersOperations.html'**, {**'data'**: data})

**userbase.html**

{%load static%}  
<!DOCTYPE **html**>  
<**html lang="en"**>  
<**title**>AutoML Template</**title**>  
<**meta charset="UTF-8"**>  
<**meta name="viewport" content="width=device-width, initial-scale=1"**>  
<**link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css"**>  
<**link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Lato"**>  
<**link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css"**>  
  
<**style**>  
body {font-family: "Lato", sans-serif}  
.mySlides {display: none}  
</**style**>  
<**body**>  
  
*<!-- Navbar -->*<**div class="w3-top"**>  
 <**div class="w3-bar w3-black w3-card"**>  
 <**a class="w3-bar-item w3-button w3-padding-large w3-hide-medium w3-hide-large w3-right" href="javascript:void(0)" onclick="myFunction()" title="Toggle Navigation Menu"**><**i class="fa fa-bars"**>Admin</**i**></**a**>  
 <**a href="{%url 'UserHome'%}" class="w3-bar-item w3-button w3-padding-large"**>Home</**a**>  
 <**a href="{%url 'UserAutoMLTest'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small"**>Auto ML</**a**>  
 <**a href="{%url 'MyPredectionsSlot1'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small"**>Data</**a**>  
 <**a href="{%url 'MyPredectionsSlot2'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small"**>Predections</**a**>  
 <**a href="{%url 'AddDataForm'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small"**>Add Data</**a**>  
 <**a href="{%url 'logout'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small"**>logout</**a**>  
 *<!--<a href="{%url 'DataUploadForm'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small">Upload</a>-->  
 <!--<a href="{%url 'DataUploadForm'%}" class="w3-bar-item w3-button w3-padding-large w3-hide-small">My Pred Upload</a>-->* <**a href="javascript:void(0)" class="w3-padding-large w3-hover-red w3-hide-small w3-right"**><**h5**> Automated Machine Learning: The New Wave of  
Machine Learning</**h5**></**a**>  
 </**div**>  
</**div**>  
  
  
  
*<!-- Page content -->*<**div class="w3-content" style="**max-width:2000px;margin-top:46px**"**>  
  
 *<!-- Automatic Slideshow Images -->  
  
  
 <!-- The Band Section -->* {%block contents%}  
  
 {%endblock%}  
  
  
*<!-- End Page Content -->*</**div**>  
  
*<!-- Image of location/map -->*<**img src="{%static 'images/map.jpg'%}" class="w3-image w3-greyscale-min" style="**width:100%**"**>  
  
*<!-- Footer -->*<**footer class="w3-container w3-padding-64 w3-center w3-opacity w3-light-grey w3-xlarge"**>  
 <**i class="fa fa-facebook-official w3-hover-opacity"**></**i**>  
 <**i class="fa fa-instagram w3-hover-opacity"**></**i**>  
 <**i class="fa fa-snapchat w3-hover-opacity"**></**i**>  
 <**i class="fa fa-pinterest-p w3-hover-opacity"**></**i**>  
 <**i class="fa fa-twitter w3-hover-opacity"**></**i**>  
 <**i class="fa fa-linkedin w3-hover-opacity"**></**i**>  
 <**p class="w3-medium"**>Powered by <**a href="#" target="\_blank"**>Alex Corporation</**a**></**p**>  
</**footer**>  
  
<**script**>  
// Automatic Slideshow - change image every 4 seconds  
var myIndex = 0;  
carousel();  
  
function carousel() {  
 var i;  
 var x = document.getElementsByClassName("mySlides");  
 for (i = 0; i < x.length; i++) {  
 x[i].style.display = "none";  
 }  
 myIndex++;  
 if (myIndex > x.length) {myIndex = 1}  
 x[myIndex-1].style.display = "block";  
 setTimeout(carousel, 4000);  
}  
  
// Used to toggle the menu on small screens when clicking on the menu button  
function myFunction() {  
 var x = document.getElementById("navDemo");  
 if (x.className.indexOf("w3-show") == -1) {  
 x.className += " w3-show";  
 } else {  
 x.className = x.className.replace(" w3-show", "");  
 }  
}  
  
// When the user clicks anywhere outside of the modal, close it  
var modal = document.getElementById('ticketModal');  
window.onclick = function(event) {  
 if (event.target == modal) {  
 modal.style.display = "none";  
 }  
}  
</**script**>  
  
</**body**>  
</**html**>

Add Data to Form

{% extends 'users/userbase.html'%}  
{%block contents%}  
 <**div class="w3-container w3-content w3-center w3-padding-64" id="band" style="**max-width:800px**"**>  
 <**h2 class="w3-wide"**>Perform Predections Analysis</**h2**>  
 <**p class="w3-opacity"**><**i**>We love Machine Learning</**i**></**p**>  
 <**p class="w3-justify"**>  
 <**center**>  
 <**form action="{%url 'AddDataToDataset'%}" method="post"**>  
 {%csrf\_token%}  
 <**label**>Enter Experience</**label**>**&nbsp;&nbsp;&nbsp;** <**input type="number" required name="Experience" min="0" max="20" step=".01"**> <**strong**>Like Training and testing Split(30,40,.....90)</**strong**>  
 <**br**/><**br**/>  
 <**label**>Enter Salary </**label**>**&nbsp;&nbsp;&nbsp;** <**input type="number" name="salary" required min="5000" max="100000" style="**size:90;**"**>  
 <**br**/><**br**/>  
 <**button type="submit" name="Test"**>Add Data</**button**>  
 </**form**>  
 <**table class="table table-bordered bg-light text-dark"**>  
 <**thead**>  
 <**tr**>  
 <**th**>S.No</**th**>  
 <**th**>Years of Experience</**th**>  
 <**th**>Salary</**th**>  
  
 </**tr**>  
 </**thead**>  
 <**tbody**>  
 {% for i in data %}  
 <**tr style="**color: Black**"**>  
 <**td**>{{forloop.counter}}</**td**>  
 <**td**>{{i.YearsExperience}}</**td**>  
 <**td**>{{i.Salary}}</**td**>  
  
  
  
 </**tr**>  
 {% endfor %}  
  
 </**tbody**>  
 </**table**>  
  
</**center**>  
 </**h2**>  
 </**form**>  
 </**p**>  
  
 <**div class="w3-row w3-padding-32"**>  
 <**div class="w3-third"**>  
 <**p**>Data Imputation</**p**>  
 <**p**>Often datasets, in reality, may contain missing values for some different reasons The randomness of MCAR data is high enough that there is no overall bias towards any particular class, unlike MAR data, which are responsible for causing an increase in bias</**p**>  
 </**div**>  
 <**div class="w3-third"**>  
 <**p**>Data Balancing</**p**>  
 <**p**>Data imbalance is a condition when one or more classes in a categorical dataset have higher observations than the rest of the classes The sample handling approach for data balancing will preprocess the training set to minimize class differences, and this issue can be resolved.</**p**>  
 </**div**>  
 <**div class="w3-third"**>  
 <**p**>Data Encoding</**p**>  
 <**p**>To make the data human-readable, the training data is often labelled in words. Data Encoding refers to converting the provided feature labels into numerical form to allow computer machines to interpret them.</**p**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
{%endblock%}

AdminLogin.html

{% extends 'base.html'%}  
{%block contents%}  
<**div class="w3-container w3-content w3-center w3-padding-64" style="**max-width:800px**" id="band"**>  
 <**h2 class="w3-wide"**>Admin Login Form</**h2**>  
 <**p class="w3-opacity"**><**i**>Be touch with us</**i**></**p**>  
 <**p class="w3-justify "**>  
 <**center**>  
 <**div class="container"**>  
 <**div class="row"**>  
 <**div class="col-sm-9 col-md-7 col-lg-5 mx-auto"**>  
 <**div class="card card-signin my-5"**>  
 <**div class="card-body"**>  
 <**h5 class="card-title text-center"**>Sign In</**h5**>  
 <**form action="{%url 'AdminLoginCheck'%}" method="post" class="form-signin"**>  
 {%csrf\_token%}  
 <**div class="form-label-group"**>  
 <**input type="text" id="inputEmail" name="loginname" class="form-control" placeholder="Enter Login Id"  
 required autofocus**>  
 <**label for="inputEmail"**>Enter Login ID</**label**>  
 </**div**>  
  
 <**div class="form-label-group"**>  
 <**input type="password" name="pswd" id="inputPassword" class="form-control"  
 placeholder="Enter Password" required**>  
 <**label for="inputPassword"**>Enter Password</**label**>  
 </**div**>  
  
 <**div class="custom-control custom-checkbox mb-3"**>  
  
  
 </**div**>  
 <**button class="btn btn-lg btn-primary btn-block text-uppercase" type="submit"**>Sign in  
 </**button**>  
 <**hr class="my-4"**>  
  
 </**form**>  
 {% if messages %}  
 {% for message in messages %}  
 <**font color='RED'**> {{ message }}</**font**>  
 {% endfor %}  
 {% endif %}  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
 </**div**>  
  
 </**center**>  
 </**p**>  
  
 <**div class="w3-row w3-padding-32"**>  
 <**div class="w3-third"**>  
 <**p**>Data Imputation</**p**>  
 <**p**>Often datasets, in reality, may contain missing values for some different reasons The randomness of MCAR  
 data is high enough that there is no overall bias towards any particular class, unlike MAR data, which  
 are responsible for causing an increase in bias</**p**>  
 </**div**>  
 <**div class="w3-third"**>  
 <**p**>Data Balancing</**p**>  
 <**p**>Data imbalance is a condition when one or more classes in a categorical dataset have higher observations  
 than the rest of the classes The sample handling approach for data balancing will preprocess the  
 training set to minimize class differences, and this issue can be resolved.</**p**>  
 </**div**>  
 <**div class="w3-third"**>  
 <**p**>Data Encoding</**p**>  
 <**p**>To make the data human-readable, the training data is often labelled in words. Data Encoding refers to  
 converting the provided feature labels into numerical form to allow computer machines to interpret  
 them.</**p**>  
 </**div**>  
 </**div**>  
</**div**>  
  
{%endblock%}