# 股评舆情分析系统

**指导教师：**

**姓名：**

**班级：**

**专业：**

目录

[股评舆情分析系统 1](#_Toc24070)

[1. 用户模块 1](#_Toc16359)

[1.1登录功能 1](#_Toc12843)

[1.1.1功能演示 1](#_Toc6931)

[1.1.2具体实现 2](#_Toc10928)

[1.2注册功能 2](#_Toc16524)

[1.2.1功能演示 2](#_Toc1494)

[1.2.2具体实现 3](#_Toc25228)

[1.3审批功能 4](#_Toc17180)

[1.4用户信息 7](#_Toc2051)

[2. 股票评论查询模块 8](#_Toc25975)

[2.1功能演示 8](#_Toc26829)

[2.2具体实现 9](#_Toc27431)

[3. 股评舆情分析模块 12](#_Toc24755)

[3.1功能演示 12](#_Toc25906)

[3.2具体实现 12](#_Toc22400)

[4. 总结 13](#_Toc13222)

## 用户模块

### 1.1登录功能

### 1.1.1功能演示

输入用户名密码，点击登录按钮，即可登录进股评舆情分析系统首页



输入用户名密码，如果用户名或密码错误，则前端页面提示错误信息



### 1.1.2具体实现

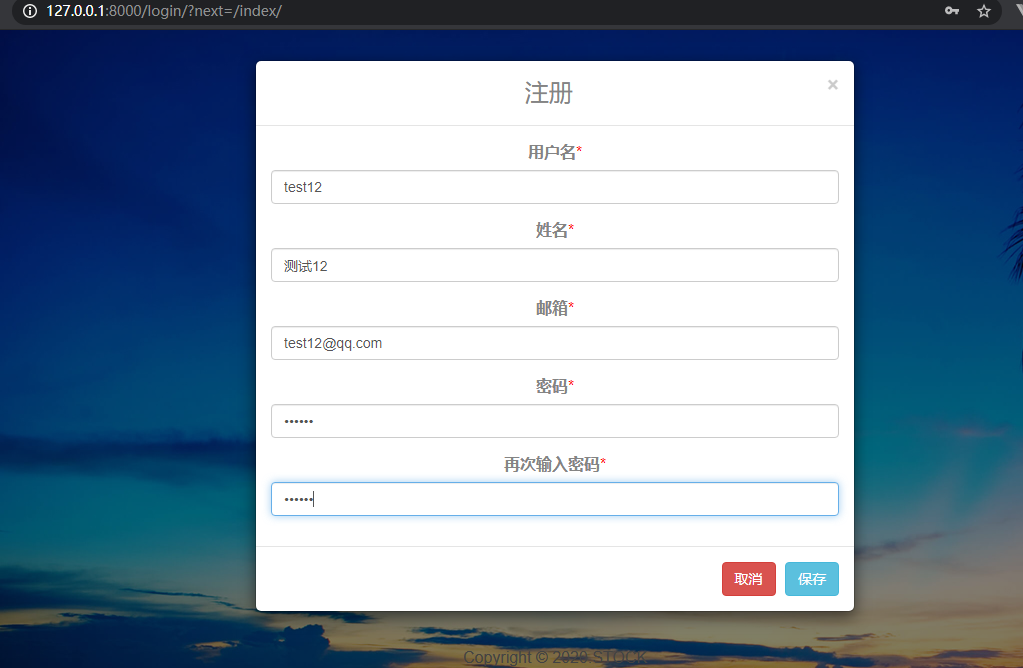
登录方法很简单，前端是表单提交。

|  |
| --- |
| @csrf\_exempt  def login\_view(request):  next\_url = request.GET.get('next', '/index/')  if request.method == "POST":  username = request.POST.get('username')  password = request.POST.get('password')  user = authenticate(username=username, password=password)  if user:  login(request, user)  return redirect(next\_url)  else:  return render(request, 'login.html', {'info': '账号或密码错误!', 'next': next\_url})  else:  return render(request, 'login.html', {'next': next\_url}) |

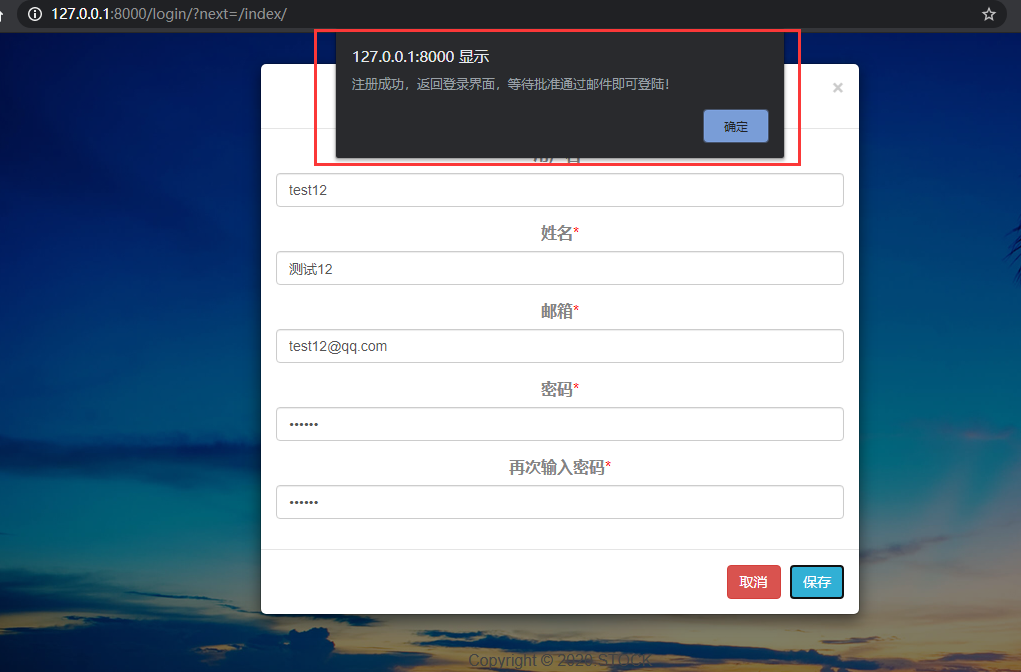
### 1.2注册功能

### 1.2.1功能演示

输入所有必填信息，用户名、真实姓名、邮箱、密码，点击保存即可



弹出注册提交成功提示，等待管理审核通过



### 1.2.2具体实现

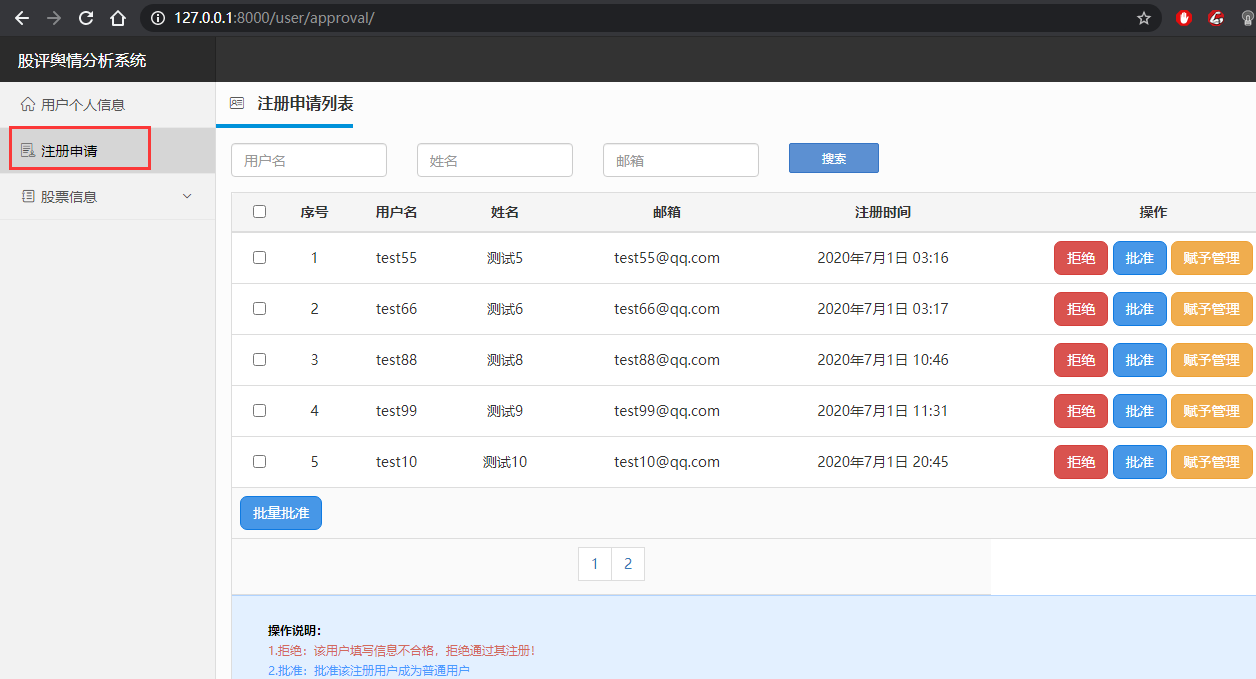
注册方法如下，前端post传过来数据，后端再次做校验，通过校验后可以提交并保存用户信息，此时用户状态是未激活的，不能登录系统，在后面管理员审核审批通过后，才会激活。

|  |
| --- |
| @csrf\_exempt  def registration(request):  if request.method == "GET":  return HttpResponse("Nothing...")  if request.method == "POST":  username = request.POST.get("username", "")  password = request.POST.get("password", "")  real\_name = request.POST.get("real\_name", "")  email = request.POST.get("email", "")  if not re.match(r'[a-zA-z]\w{5,15}', username):  return HttpResponse(content=json.dumps({"result": "请正确填写用户名（以字母开头）!"}))  if not re.match(r'^[0-9a-zA-Z\_]{0,19}@[0-9a-zA-Z]{1,13}\.[com,cn,net]{1,3}$', email):  return HttpResponse(content=json.dumps({"result": "请正确填写用户邮箱!"}))  if username and password and real\_name and email:  is\_username\_existed = User.objects.filter(username=username)  is\_email\_existed = User.objects.filter(email=email)  if is\_username\_existed:  return HttpResponse(content=json.dumps({"result": "用户名已存在,请使用其他用户名!"}))  elif is\_email\_existed:  return HttpResponse(content=json.dumps({"result": "邮箱已被注册,请更换邮箱注册!"}))  try:  user = User.objects.create\_user(username=username,  password=password,  email=email,  is\_active=0)  UserProfile.objects.create(user\_real\_name=real\_name,  auth\_user=user,  is\_approval=0,  is\_manager=0) # 提交注册给未处理状态  except Exception as e:  return HttpResponse(content=json.dumps({"result": e}), status=400)  return HttpResponse(content="success")  else:  return HttpResponse(content=json.dumps({"result": "必填项不能为空!"})) |

### 1.3审批功能

#### 1.3.1功能演示

登录系统后，点击注册申请菜单即可查询出待审批用户列表



对某一用户点击蓝色按钮即可审批通过，此时该用户成为普通用户，仅能看到用户个人信息菜单；

对某一用户点击黄色按钮即可审批通过，此时该用户成为管理员用户，能看到所有菜单；

对某一用户点击红色按钮即可拒绝审批通过，此时该用户成为无效用户，不可登录系统；

#### 1.3.2具体实现

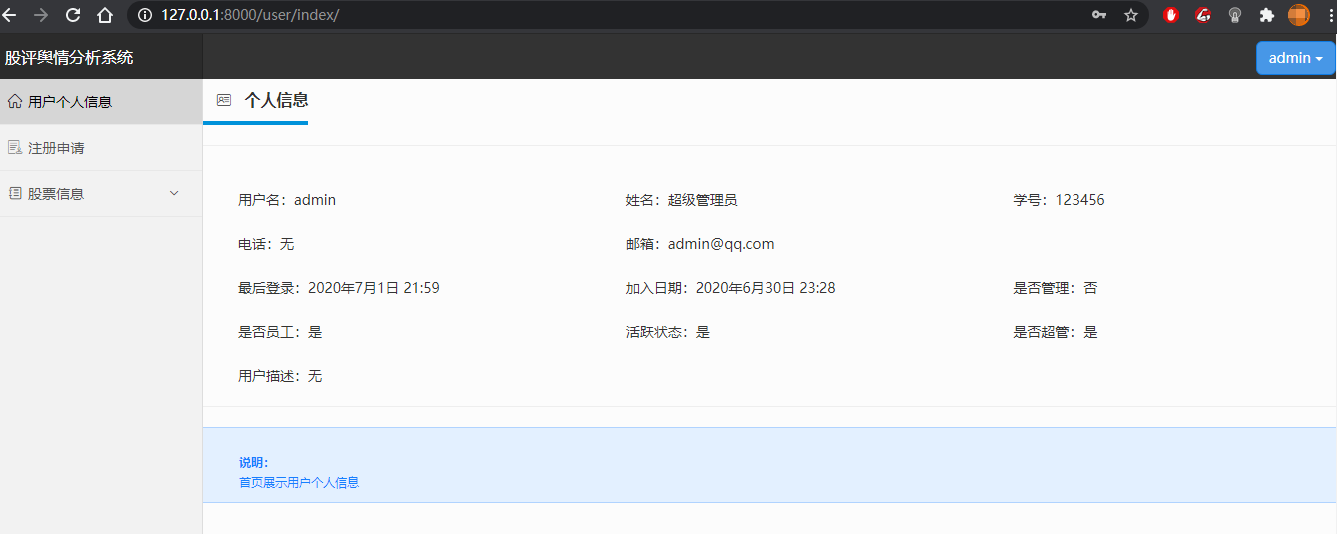
以下四个方法registration\_dispatch为后端查询待审批用户方法，reject\_user为拒绝用户审批通过方法，pass\_user和pass\_manager\_user分别为通过用户注册成为普通和管理员用户方法。

后面三个方法作用类似，查询搜索方法registration\_dispatch 支持多条件模糊查询，并实现自定义分页方法，后面股票评论信息模块可以看到分页方法的强大。

|  |
| --- |
| @login\_required(login\_url="/login/")  def registration\_dispatch(request):  from apps.Information.views import paginator\_interface  username = request.GET.get("username", "")  user\_real\_name = request.GET.get("user\_real\_name", "")  email = request.GET.get("email", "")  page = int(request.GET.get("page", 1))  keys = "" # 搜索参数  search = dict()  search["is\_approval"] = 0  if username:  search["auth\_user\_\_username\_\_contains"] = username  keys += "username=" + username + "&"  if user\_real\_name:  search["user\_real\_name\_\_contains"] = user\_real\_name  keys += "user\_real\_name=" + user\_real\_name + "&"  if email:  search["auth\_user\_\_email\_\_contains"] = email  keys += "email=" + email + "&"  profile = UserProfile.objects.filter(\*\*search).order\_by('user\_id')  # 调用自定义分页方法  zip\_obj, pages, page\_obj, page, paginator = paginator\_interface(profile, page, TABLE\_SIZE)  return render(request, 'approval.html', {"page\_obj": page\_obj, 'zip\_obj': zip\_obj, "keys": keys,  'page': page, 'paginator': paginator, 'pages': pages})  @csrf\_exempt  @login\_required(login\_url="/login/")  @transaction.atomic  def reject\_user(request):  if request.method == "GET":  return HttpResponse("Nothing...")  if request.method == "POST":  username = request.POST.get("username", "")  user\_profile = UserProfile.objects.get(auth\_user\_\_username=username)  user\_profile.is\_approval = -2 # 拒绝注册，比停用状态低一级别  user\_profile.save()  return HttpResponse(content="success")  @csrf\_exempt  @login\_required(login\_url="/login/")  @transaction.atomic  def pass\_user(request):  if request.method == "GET":  return HttpResponse("Nothing...")  if request.method == "POST":  username\_list = json.loads(request.POST.get("username\_list", []))  for username in username\_list:  user\_profile = UserProfile.objects.get(auth\_user\_\_username=username)  user\_profile.is\_approval = 1 # 正常用户  user\_profile.auth\_user.is\_active = 1 # 活动帐户，不决定能否登陆  user\_profile.save()  user\_profile.auth\_user.save()  user\_profile.save()  return HttpResponse(content="success")  @csrf\_exempt  @login\_required(login\_url="/login/")  @transaction.atomic  def pass\_manager\_user(request):  if request.method == "GET":  return HttpResponse("Nothing...")  if request.method == "POST":  username = request.POST.get('username', '')  user\_real\_name = request.POST.get('user\_real\_name', '')  email = request.POST.get('email', '')  user\_profile = UserProfile.objects.get(auth\_user\_\_username=username)  user\_profile.auth\_user.is\_active = 1 # 注册通过  user\_profile.is\_approval = 1 # 正常用户  user\_profile.is\_manager = 1 # 是管理  user\_profile.save()  user\_profile = UserProfile.objects.get(auth\_user\_\_username=username)  user\_profile.save()  return HttpResponse(content="success") |

## 1.4用户信息

#### 1.4.1功能演示



#### 1.4.2具体实现

后端：用户模块拓展Django的用户表，用try...except...语句将UserProfile表数据查询出来返回给页面，如果该用户是超级管理员，则回传特殊信息。

|  |
| --- |
| @login\_required(login\_url="/login/")  def user\_profile\_view(request):  user = User.objects.get(username=request.user.username)  try:  user\_profile = UserProfile.objects.get(auth\_user=user)  except Exception as e:  if user.is\_superuser:  user\_profile = {  "user\_real\_name": "超级管理员",  "email": user.email,  "user\_stu\_num": "123456",  }  else:  return redirect("/login/")  return render(request, 'index.html',  {"user": user, "user\_profile": user\_profile, }) |

前端用django自带的模板标签语言将回传的数据渲染出来。

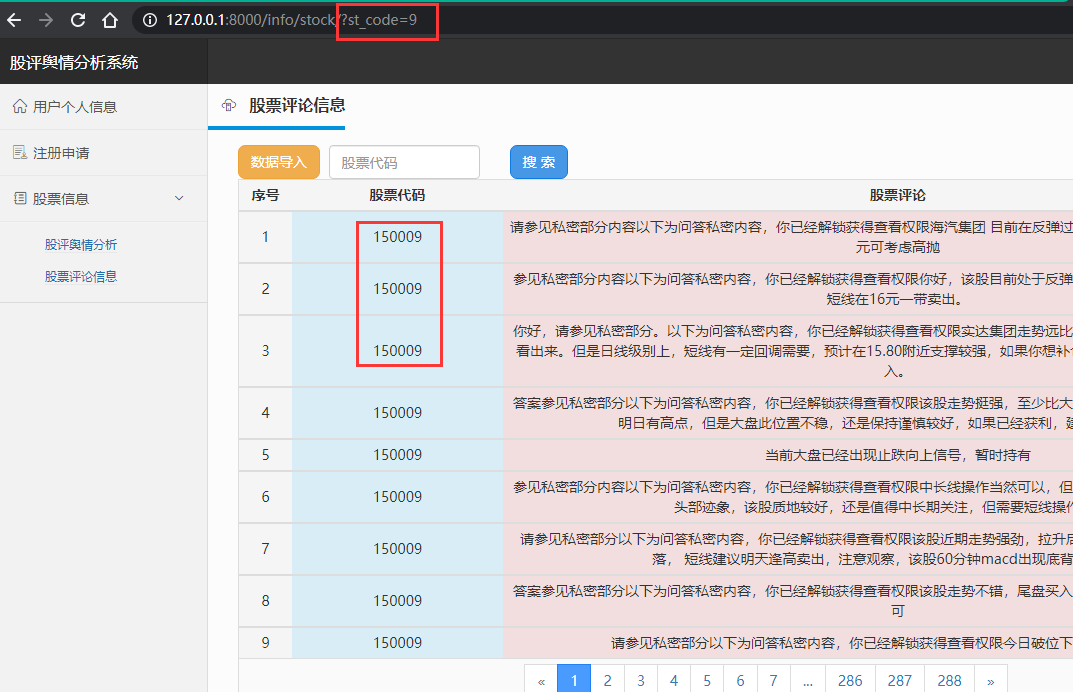
## 股票评论查询模块

### 2.1功能演示

点击股票评论信息菜单即可看到所有的股票代码以及对应的评论信息

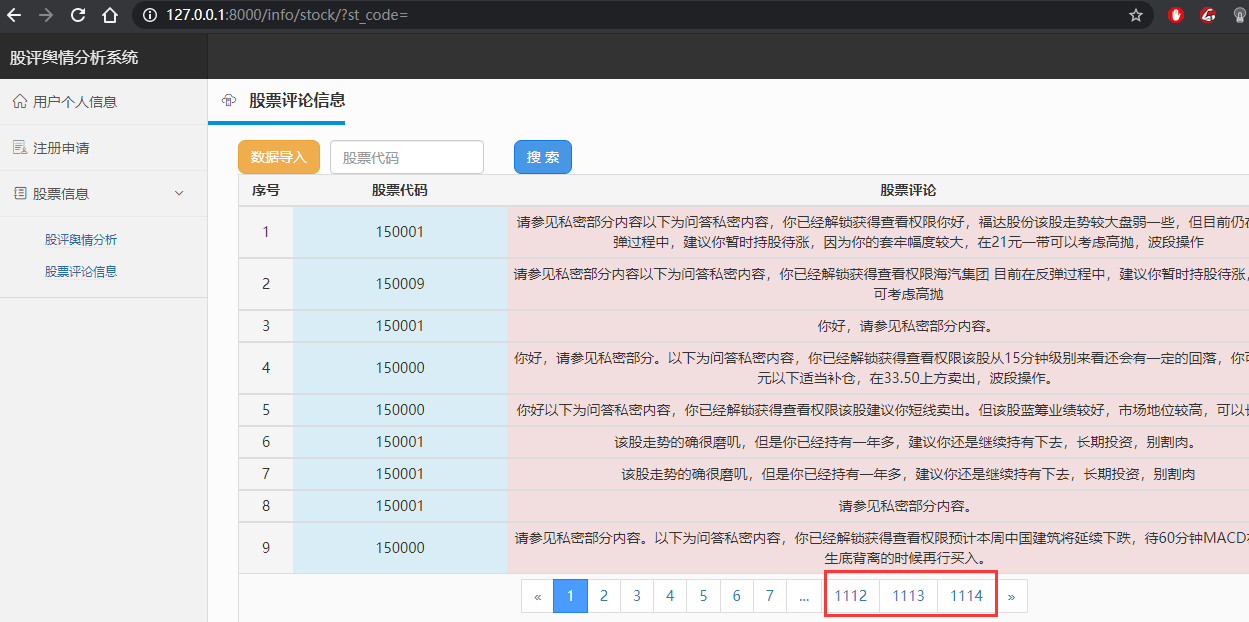


此页面支持用股票代码模糊查询出相对应的股票以及评论信息，如下图所示，输入9即可搜索出股票代码包含数字9的股票信息



点击数据导入按钮可将csv文件中的数据导入数据库中，刷新后即可发现股票信息的页数多了五百多页

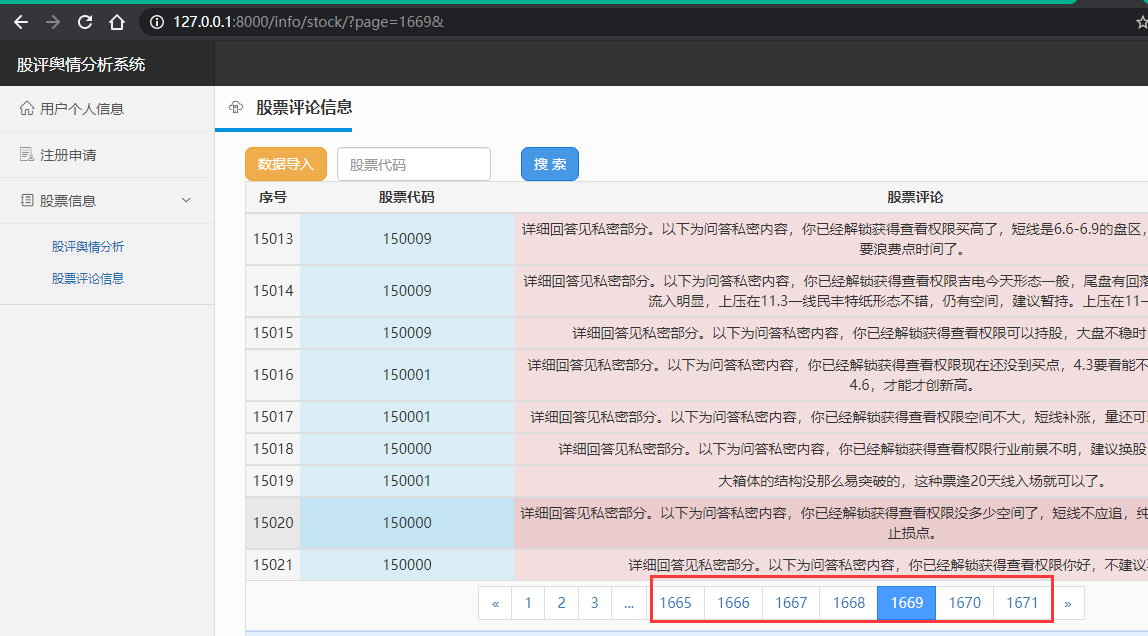
导入前1114页



点击导入按钮提示导入成功



导入后页数为1671页



### 2.2具体实现

* paginator\_interface方法为分页方法，从数据库查询到数据后，需要用到分页的方法调用分页方法返回给页面即可，页面通过django的模板标签语言渲染后，可完成分页效果。
* read\_csv方法为使用pandas读取csv文件并进行数据的清洗处理
* import\_csv\_view方法将数据做进一步的清洗并返回给前端页面

|  |
| --- |
| @csrf\_exempt  @transaction.atomic  @login\_required(login\_url="/login/")  def stock\_stat\_view(request):  st\_code = request.GET.get("st\_code", "")  page = int(request.GET.get("page", 1))  keys = "" # 搜索参数  search = dict()  if st\_code:  search["st\_code\_\_contains"] = st\_code  keys += "st\_code=" + st\_code + "&"  stock\_stat = StockStat.objects.filter(\*\*search).order\_by('-st\_id') # 根据st\_id倒序  # 调用自定义分页方法  zip\_obj, pages, page\_obj, page, paginator = paginator\_interface(stock\_stat, page, TABLE\_SIZE)  return render(request, 'stock\_stat.html', {"page\_obj": page\_obj, 'zip\_obj': zip\_obj, "keys": keys,  'page': page, 'paginator': paginator, 'pages': pages})  def read\_csv():  '''读取csv文件  '''  df = pd.read\_csv(src\_file, encoding='GBK', error\_bad\_lines=False) # 读取源文件  row, col = df.shape  contents\_list = df.iloc[:, 0].values.tolist() # 数据列表  return contents\_list  @csrf\_exempt  @transaction.atomic  @login\_required(login\_url="/login/")  def import\_csv\_view(request):  contents\_list = read\_csv()  clear\_contents\_list = [x.strip() for x in contents\_list if x.strip() != ''] # 去换行符和空格  querysetlist = [] # 批量存储准备  for desc in clear\_contents\_list:  desc\_num = desc[0] # 每行第一个数字  desc\_str\_base = re.findall(r'\d{1}(.\*)$', desc)[0]  desc\_str = desc\_str\_base.lstrip().rstrip() # 评论  querysetlist.append(StockStat(st\_code="15000" + desc\_num, st\_desc=desc\_str))  res = StockStat.objects.bulk\_create(querysetlist) # 批量存储  if res:  return HttpResponse(content="success")  else:  return HttpResponse(content=json.dumps({"result": "导入csv失败!请检查表格文件"}))  def paginator\_interface(object\_list, page, table\_size):  """  分页接口函数，创建分页的规则，返回给函数表格序号，页数对象  :param object\_list: 搜索结果列表  :param page: 前端返回的页数  :param table\_size: 表格长度  :return:  """  paginator = Paginator(object\_list, table\_size)  try:  page\_obj = paginator.page(page)  except PageNotAnInteger:  # 如果请求的页数不是整数, 返回第一页。  page\_obj = paginator.page(1)  except EmptyPage:  # 如果请求的页数不在合法的页数范围内，返回结果的最后一页。  page\_obj = paginator.page(paginator.num\_pages)  except InvalidPage:  return HttpResponseRedirect(next) # 如果请求的页数不存在, 重定向页面  index = [table\_size \* (page - 1) + x for x in [x + 1 for x in list(range(table\_size))]] # 页内列表序号  zip\_obj = zip(page\_obj, index)  # 页数控制  num\_pages = paginator.num\_pages  if num\_pages <= 11: # 总页数小于11  pages = [x for x in range(1, min(num\_pages + 1, 12))]  elif num\_pages > 11 and page <= 6: # 总页数大于11，激活页面为前6页  pages = [x for x in range(1, min(num\_pages + 1, 8))] + [None] + list(range(num\_pages + 1))[-3:]  elif page > num\_pages - 6: # 总页数大于11，激活页面为后6页  pages = list(range(4))[1:] + [None] + [x for x in range(num\_pages - 6, num\_pages + 1)]  else: # 总页数大于11，激活页面为中间其他页  pages\_front = [x for x in range(1, min(num\_pages + 1, 4))] # 前3  pages\_behind = list(range(num\_pages + 1))[-3:] # 后3  pages\_mid = [page - 1, page, page + 1] # 中3  pages = pages\_front + [None] + pages\_mid + [None] + pages\_behind  return zip\_obj, pages, page\_obj, page, paginator |

## 股评舆情分析模块

### 3.1功能演示



输入股票评论并点击蓝色提交预测按钮，即可在下方得到预测的结果



预测结果如下



### 3.2具体实现

通过输入的评论进行自然语言的分析及模型的过滤，返回给前端，前端django自带的模板标签语言渲染后给用户呈现结果。

|  |
| --- |
| @csrf\_exempt  @transaction.atomic  @login\_required(login\_url="/login/")  def predict\_stock(request):  if request.method == 'GET':  st\_desc = request.GET.get('st\_desc', '')  if st\_desc:  encode = [1012,6442,347,6237,7219,3389,4692,3326,7361,809,678,3221,4900,2166,1079,2159,64215]  pred = 5  if pred == 0:  pred\_res = '看跌'  elif pred == 5:  pred\_res = '看涨'  else:  pred\_res = '不能确定'  res = dict()  regex\_str = ".\*?([\u4E00-\u9FA5]+).\*?" # 匹配中文正则表达式  match\_obj = re.findall(regex\_str, st\_desc)  if match\_obj:  clear\_st\_desc = ''.join(match\_obj)  res['clean\_st\_desc'] = clear\_st\_desc  res['encode'] = encode  res['pred\_res'] = pred\_res  return render(request, 'predict\_stock.html', {'res': res})  else:  res = dict()  res['clean\_st\_desc'] = "无"  res['encode'] = "无"  res['pred\_res'] = "无"  return render(request, 'predict\_stock.html', {'res': res})  else:  res = dict()  res['clean\_st\_desc'] = "无"  res['encode'] = "无"  res['pred\_res'] = "无"  return render(request, 'predict\_stock.html', {'res': res})  else:  res = dict()  res['clean\_st\_desc'] = "无"  res['encode'] = "无"  res['pred\_res'] = "无"  return render(request, 'predict\_stock.html', {'res': res}) |

## 总结

通过编写股评舆情分析系统让我们系统的学习了web系统的工作原理，理清楚了前端后端协作的流程，并结合了pandas、jieba等数据处理、自然语言处理的工具，让我们领略了python语言的强大，激发了学习的兴趣。