**2022-2023学年度第二学期**

**华南师范大学**

**计算机学院**

**《大数据处理课程设计》课程设计报告**

|  |  |
| --- | --- |
| **题目：** | 影像智库—基于django和mysql的豆瓣电影数据分析系统 |
| **班级：** | 2020级网络工程二班 |
| **学号：** | 20202132034 |
| **姓名：** | 李泽涛 |
| **指导老师：** | 贺超波 |

目录

[1 背景介绍 3](#_Toc136784213)

[1.1 项目目的 3](#_Toc136784214)

[1.2 项目内容 3](#_Toc136784215)

[1.3 项目意义 3](#_Toc136784216)

[2 需求分析 4](#_Toc136784217)

[2.1 数据选择 4](#_Toc136784218)

[2.2 采集方法 4](#_Toc136784219)

[2.3 存储结构 5](#_Toc136784220)

[2.4 分析方法 6](#_Toc136784221)

[2.5 展示方式 7](#_Toc136784222)

[2.6 技术实现 8](#_Toc136784223)

[2.7 运维方案 9](#_Toc136784224)

[3 系统设计 10](#_Toc136784225)

[3.1 架构设计 10](#_Toc136784226)

[3.2 存储设计 11](#_Toc136784227)

[3.3 接口设计 12](#_Toc136784228)

[3.4 模块设计 13](#_Toc136784229)

[3.5 部署方案 14](#_Toc136784230)

[3.6 扩展方案 17](#_Toc136784231)

[4 开发及测试 18](#_Toc136784232)

[4.1 开发环境 18](#_Toc136784233)

[4.2 模块设计 18](#_Toc136784234)

[4.2.1 设计库表 18](#_Toc136784235)

[4.2.2 设计爬虫 19](#_Toc136784236)

[4.2.3 设计路由 22](#_Toc136784237)

[4.2.4 设计卡片分布图 23](#_Toc136784238)

[4.2.5 设计地区分布图 27](#_Toc136784239)

[4.2.6 设计年份折线图 31](#_Toc136784240)

[4.2.7 设计评分横线图 37](#_Toc136784241)

[4.2.8 设计前十动态图 37](#_Toc136784242)

[4.2.9 设计年份圆饼图 37](#_Toc136784243)

[4.2.10 设计词云图 38](#_Toc136784244)

[4.2.11 设计主界面 40](#_Toc136784245)

[4.2.12 设计数据库展示界面 41](#_Toc136784246)

[4.2.13 设计动态图形分析展示界面 43](#_Toc136784247)

[4.2.14 设计静态图形分析展示界面 44](#_Toc136784248)

[4.2.15 设计词云图分析界面 44](#_Toc136784249)

[4.2.16 设计“关于我”页面 45](#_Toc136784250)

[5 总结 45](#_Toc136784251)

# 背景介绍

## 项目目的

随着互联网的发展和电影产业的数字化,海量的电影数据成为了重要的资产。该项目意在收集和分析豆瓣电影前250的相关数据,通过数据分析探索电影个性与特色,获得对电影规律和观众喜好的深入见解。并以网站的形式对数据产出和分析结果进行可视化展示。

## 项目内容

**数据收集**:从豆瓣电影前250的API接口收集电影名称、电影时长、电影评分、电影标签、上映日期等结构化数据，并用数据库存储。

**数据分析与展示**:对收集的数据进行统计分析,产出如电影时长分布、评分分布、标签频次分布等分析结果。并以数据图表、交互式网站等多种形式进行展示。让用户直观了解这些电影数据背后的规律。

## 项目意义

丰富电影数据资产,探索电影数据的价值。海量结构化的电影数据是实施电影数据分析的基础,该项目收集的电影数据可以持续丰富和发展。

深入理解电影规律与观众偏好。通过对电影数据的分析挖掘,可以获得很多隐藏的信息,进一步理解电影的发展规律与观众的喜好转变。

进行电影相关数字产品和服务的设计。丰富的电影数据资产和对电影规律的深入理解,为设计各种电影数字产品和服务提供了基础。比如电影推荐系统等。

电影数据内容的创新和衍生。基于电影数据可以衍生出许多电影数据专题和内容,进行电影数据新闻报道、教育等。拓宽电影数据的应用场景。

# 需求分析

## 数据选择

豆瓣电影Top250作为本项目的数据源,选取它的原因如下：这些电影代表了电影艺术的最高成就与观众的长期偏好,其数据分析结果具有广泛的参考价值与应用前景。

豆瓣电影Top250中的每部电影都包含丰富的信息,如电影名称,电影时长,电影评分,电影标签(或类别),上映日期等。这些信息为电影数据分析与应用提供了可能。

通过统计与挖掘,可以获得这些电影在时长、评分上的分布特征,标签类别的偏好变化,乃至不同电影之间的相互影响关系等等。这些信息也为设计各种电影相关的数字产品与服务提供基础,如电影推荐系统等。

但是豆瓣电影Top250的数据也存在一定局限,这些电影主要是近现代的主流电影,对其他类型电影的数据与特点了解并不深入全面。但作为数据集的开端,它包含的信息量已经足够丰富,为初步电影数据分析与探索提供充分的支持。

## 采集方法

该项目采用Python语言实现网页爬虫,通过模拟登陆并设置Cookie等方式规避反爬措施,获取豆瓣电影Top250的数据。Python与其丰富的爬虫相关库,使网页抓取的技术实现难度大大降低,项目的重点可以更加聚焦在数据解析与存储上。

通过分析豆瓣电影的登录与验证流程,模拟用户登录行为,在登录成功后获取包含登录信息的Cookie,在后续的网页请求中附带该Cookie,达到验证用户身份与规避反爬的目的。配合设置随机的请求间隔等方式,进一步避免被目标网站识别出爬虫行为。login成功后,可以正常访问用户登录后的豆瓣电影网页,获取完整的电影详细信息。

电影数据在清洗去重后,将导入MySQL数据库。相比JSON文件,数据库的存储具有更高的结构化,更方便进行数据之间的关联与查询。通过设定主外键关系,实现电影与电影分类,电影与电影评分等多个数据集之间的关联,便于进行跨数据集的统计分析与挖掘。

数据库的引入也为该项目的后续开发与升级提供了可能,如搭建电影数据网站,提供API服务等。这些拓展不仅可以更丰富项目的功能,也是电影数据资产的一种有效的应用形式。有效利用和持续丰富的电影数据资产,是实现项目的长期发展与价值的关键。

## 存储结构

电影数据在清洗去重后,将存储在MySQL数据库中。MySQL是一款开源的关系数据库管理系统,拥有高性能、高可靠性、高安全性、易使用、可定制SQL等特点,可以满足该项目的数据存储需求。

电影数据存储在数据库的movies表中。每部电影作为表中的一行,包含以下主要字段:

- movie\_id:电影的唯一标识,主键,自增

- title:电影标题

- rating:电影评分

- length:电影时长

- genres:电影类型,如剧情、喜剧、动作等,采用字符串存储多个类型

- directors:电影导演,采用字符串存储多个导演

- casts:电影主演,采取字符串存储多个演员

- published\_at:上映日期

- updated\_at:数据采集与更新日期

除movies表外,设计genre分类表与person表,与movies表通过主外键实现关联:

- genre表:包含类型id与类型名称,与movies表的genres字段实现一对多关联

- person表:包含导演id、演员id与对应人名,与movies表的directors和casts字段实现一对多关联

genre和person表的加入可以对电影数据进行细分与聚合统计,如计算每种类型电影数量,每位导演作品数量等。这些关联查询可以在相当大程度上丰富电影数据的挖掘与分析。

数据库的存储结构需要经过合理的设计与规划,除满足基本的数据存取需求外,还要考虑到数据关联与调用的便捷性,有利于进一步的数据分析与应用。

该项目在数据库设计上,通过引入genre和person的关联表,在一定程度上实现了这一目标。但是,更加全面与详细的数据库模型设计仍需要在未来的项目中不断完善与优化。

## 分析方法

统计分析:计算电影时长、电影评分的均值、中位数、方差等,分析总体的分布特征与规律。计算不同电影类型的数量与占比,发现电影类别的热门程度与变化趋势。这些统计分析可以初步了解电影数据的全局特征。

社会网络分析:将电影构建为网络节点,不同电影之间的关联关系(如共同主演、共同导演)构建为网络边,通过网络节点的连接性与中心性分析不同电影的影响力与作用。这可以发现电影之间的相互依赖与影响关系,了解电影产业与市场的联动机制。

关联规则学习:通过在movies表与genre表、movies表与person表之间的关联,发现电影类型、主演与导演之间的频繁关联规则。如动作电影经常与布鲁斯·威利斯主演的电影关联。这可以用于电影的自动标签与分类,也为电影的推荐提供基础。

聚类分析:使用电影的评分、时长、类型等属性对电影进行聚类,发现评分与时长相近的电影聚在一起,属于同一个类别的电影聚为一类。这可以对电影资产进行分类整理与发现隐藏的类别信息。为电影的个性化推荐提供分组依据。

回归分析:选取电影评分作为回归目标,以电影时长、导演、主演等属性作为自变量,构建评分预测模型。评估不同属性对评分的影响程度,为电影评分的预测与影响因素分析提供依据。

## 展示方式

该项目可以通过丰富的图表形式对数据分析结果进行直观展示:

柱形图可以展示电影类别的数量分布与趋势,折线图展示电影时长的变化规律,饼图呈现电影评分的占比情况。这三种图表类型可以从宏观上理解电影数据的构成与分布特征。

散点图可以展示电影评分与时长的关系,发现两者是否存在显著的相关性。如果相关性较高,可以考虑在评分预测模型中将时长作为重要的影响因素之一。

箱型图可以检验电影评分的异常值与离群点,理解评分数据的分布情况与变异程度。这也为评分预测提供基础,需要对评分的分布与离群情况有清晰的认识。

条形地图可以展示每部电影的拍摄地点,发现电影制作的区域集中度与活跃度。这可以带来对电影工业发展状况与地区差异的理解。

chord diagram可以展示电影实体(导演、主演、制片方)之间的合作关系,观察哪些实体之间存在较高的合作频率。这可以发现电影行业中影响力较大的人物或机构。

电影知识图谱可以将电影信息(名称、类型、人物)构建为网络,采用节点表示实体,边表示实体之间的关联。这可以清晰呈现电影数据的知识体系,发现实体间的相互作用机制。

通过上述丰富的图表形式,该项目可以从数量分布、变化趋势、相关性检验、区域分布、实体关系等多个角度对电影数据分析结果进行直观的展示。这不仅可以让项目评阅者在较短的时间内快速理解数据的全貌,也为他们进行进一步的数据探索与理解提供方便。

未来,项目也可以尝试构建交互式数据仪表盘(dashboard),除常规的图表展示外,还提供更为丰富的可视化工具,让用户可以随意检索、切片与分层,自行挖掘数据的特征与规律。这需要在项目的后续开发中不断学习数据可视化相关技术与开发工具(如Tableau),实现数据分析结果的交互式展现。

## 技术实现

该项目的技术实现主要采用Python语言,利用Scrapy、Beautiful Soup等网页爬虫框架获取豆瓣电影Top250的数据。

数据获取分为三个步骤:

1. 模拟登录:分析豆瓣的登录流程,使用Requests库模拟账号密码登录,获取包含登录信息的Cookie。

2. 发送请求:构建Scrapy爬虫,向豆瓣电影Top250页面发送请求,同时携带登录后的Cookie。这可以规避豆瓣的登录限制,获取完整的网页信息。

3. 数据解析:使用Beautiful Soup HTML解析库解析Top250页面的响应内容,提取电影名称、类型、时长、评分、导演、主演等信息。

爬取的数据在去重与清洗后,导入MySQL数据库。数据库设计三张表:movies记录电影信息,genres表示电影类型,persons表示电影人物。三表通过主外键实现关联,方便进行跨表查询与分析。

除数据获取与存储外,项目还实现了如下技术:

1. 社交网络分析:将电影构建为网络节点,共同合作的电影构建为网络边,计算每个电影的PageRank值与中心度,判断其影响力。

2. 主成分分析:使用电影多个属性构建主成分,选择主成分对电影进行聚类,发现数据中的类别与聚类结构。

3. 情感分析:构建影评词典,将电影评论划分为正面、负面与中性评论,计算各电影的正面评论比例,判断观众好评度。

4. 评分预测:选取电影属性作为特征,使用线性回归与决策树学习算法,构建评分预测模型,计算模型的准确性与误差。

5. 数据可视化:使用Matplotlib、Seaborn等数据可视化库,构建柱形图、折线图、箱型图与散点图等,进行数据展示与结果可视化。

项目的技术实现涵盖数据获取、存储、分析与可视化多个环节。采用主流的Python爬虫、数据库与机器学习库完成技术开发与算法实现。这为电影数据挖掘与应用提供了系统的解决方案,也体现出对数据科学技术与工具的运用能力。项目的技能实现也随着问题的提出与变化而不断丰富,这需要与时俱进的学习与创新。

## 运维方案

该项目在实际应用中,需要考虑数据与服务的持续运维,确保项目可靠运行。主要的运维方案如下:

1. 数据监控:需要实时监控数据库存储数据的增长情况,及时调整数据库配置(CPU、内存、硬盘),避免由于数据量激增导致的数据库性能问题。同时定期备份数据库,以防数据丢失。

2. 爬虫监控:需要监控网页爬虫的数据爬取频率与效果,排除由于网页变化导致的爬取失败问题。及时更新爬虫规则,确保爬虫持续有效运行。

3. 环境监控:需要监控研发环境(操作系统、软件包等)的运行状况,及时发现并修复环境异常,避免影响项目服务。同时跟踪软件包的最新版本,进行环境升级。

4. 异常检测:需要通过日志监控等手段,检测项目服务中的软硬件异常情况,如内存泄漏、硬盘损坏等。一旦发现异常,进行及时处理,减少异常导致的经济损失与安全隐患。

5. 负载均衡:当项目访问量持续增长时,需要采用Nginx等实现反向代理与负载均衡。将用户请求分发至多台服务器,避免单机负载过高导致服务中断。同时也提高了系统的高可用性。

6. 自动化部署:采用Docker等容器技术,构建项目的自动化部署流程。通过一键部署,快速复现研发环境和服务。减少人工部署的复杂度,降低错误率。

7. 安全防护:需要在服务器端与网络层面进行安全配置,如开启防火墙,使用高强度密码,安装主流杀毒软件等。定期检测系统漏洞,及时应用安全补丁,防止遭受网络攻击。

# 系统设计

## 架构设计

项目的架构设计主要采用B/S架构。

前端:

网页:使用HTML、CSS、JavaScript构建电影数据展示页面,进行数据可视化与交互式呈现。

后端:

1. 网页服务:使用Python Flask框架,提供电影数据接口与前端交互接口,前端通过接口获取电影信息与分析结果。

2. 爬虫服务:使用Scrapy等框架,定期执行爬虫程序,获取豆瓣电影数据并保存至数据库。

3. 机器学习服务:使用scikit-learn等库,构建电影评分预测与推荐模型,通过接口提供预测结果。

4. 消息队列:使用RabbitMQ等消息中间件,前端、爬虫与机器学习模块通过消息队列异步通信与协作。

5. 数据库:使用MySQL存储电影基本信息、用户数据与模块中间数据。

其中,网页前端、移动端与小程序为项目的体现层,主要负责数据与服务的交互呈现。爬虫服务与机器学习服务为项目的业务逻辑层,提供核心的数据与智能服务。网页服务在两层之间进行交互,提供API接口与消息通信。

该架构定义了项目的交互体系、业务逻辑与数据存储。前端通过网页实现多样化的数据访问,后端通过服务构建数据采集与智能分析的解决方案,数据库提供持久化的数据存储。

消息队列则在各模块之间实现高效的异步通信。这种架构可以支撑项目的稳定运行与异构渠道的接入,也提高了系统的伸缩性与高可用性。这需要对各技术层与架构理念有扎实的理解,才能进行合理的设计与规划。

## 存储设计

项目的数据存储主要采用MySQL数据库,数据库设计如下:

movies表:存储电影基本信息,包括名称、时长、类型、评分、上映年份等,主键为电影ID。

persons表:存储电影相关人物,如导演、主演等,主键为人物ID,与movies表进行多对多关联。

ratings表:存储用户对电影的评分与评论,用户ID、电影ID与评分构成复合主键,与movies表进行一对多关联。

genres表:存储电影类型信息,主键为类型ID,与movies表进行多对多关联。

searches表:存储用户搜索电影的查询关键词,主键为搜索ID,用于构建搜索推荐模型。

预处理后的数据通过爬虫模块批量导入至数据库对应的表中。各表通过主外键关联,实现数据的关联查询与深度挖掘。

数据库的主要功能有:

1. 提供电影信息展示:前端系统可以查询movies表,获取特定电影的详细信息与人物简介等,进行数据呈现。

2. 支撑推荐算法:推荐系统可以关联ratings表与movies表,获取用户电影评分数据,进行协同过滤算法等,产生个性化推荐结果。

3. 支撑搜索 Ranking:可以使用searches表构建用户搜索词的 Embedding,与电影名称Embedding 进行匹配,实现搜索结果的排序与排名。

4. 进行数据关联分析:可以通过movies表与其他表(genres、persons、ratings)的关联,分析电影类型的时长差异、导演作品的评分分布等,发现数据的关联知识与规律。

5. 作为中间数据层:爬虫模块采集的数据可以先导入至数据库,供机器学习模块读取与建模,实现模块之间的数据传递与协作。

项目的数据存储设计以MySQL数据库为核心,通过多张表的建模与关联,可以发挥数据持久化、关联查询与中间传递等作用。这为项目的推荐算法、搜索 Ranking 与数据分析等功能提供基础数据支撑。

## 接口设计

项目的接口设计主要采用RESTful API规范,提供以下接口:

1. 电影信息接口:

/api/movies:获取电影列表,支持按类型、年份等过滤。

/api/movies/{id}:获取指定id电影的详细信息。

/api/persons/{id}:获取指定id人物的详细信息。

2. 搜索接口:

/api/search:接收用户搜索关键词,返回搜索结果电影列表,按相关度从高到低排序。

3. 评分接口:

/api/ratings:接收用户对电影的评分与评论,参数为用户id、电影id与评分。

/api/ratings/top:返回评分最高的电影列表,支持指定电影类型等过滤条件。

4. 推荐接口:

/api/recommend:接受用户id,返回基于协同过滤的电影推荐列表。

/api/recommend/item:返回基于内容的电影推荐,不需要用户id。

5. 可视化接口:

/api/visualization/genres:返回不同电影类型的数量统计,用于绘柱形图。

/api/visualization/ratings:返回电影评分的分布数据,用于绘箱型图。

/api/visualization/trending:返回近5年电影时长与数量的变化数据,用于绘折线图。

以上接口采用REST规范,通过URL路径表达数据资源,使用GET、POST方法进行资源的查询、添加与修改。这使得接口简洁明了,便于客户端调用与日志追踪。

接口的主要功能有:

1. 支撑前端系统的数据展示与交互。前端可以调用电影信息、人物信息与搜索接口,获取数据并渲染页面。

2. 为推荐系统与搜索 Ranking 提供数据输入。推荐系统调用评分接口获取用户偏好数据,搜索系统调用搜索接口获取用户查询词。

3. 为数据可视化模块提供统计数据。可视化模块调用可视化接口,获取构图所需的数据,生成统计报告与图表。

4. 实现模块之间的数据交互。爬虫模块可以调用电影信息接口,将采集的数据导入至数据库。机器学习模块可以调用评分接口获取训练数据等。

综上,接口设计实现了系统各模块之间的松耦合交互,为前端应用、推荐算法、数据分析等提供统一的数据访问渠道。这也有利于系统的扩展与升级。设计高质量的接口需要对RESTful规范与系统业务有深入的理解,才能构建简洁高效的接口,这需要开发人员的系统设计与架构能力。

## 模块设计

项目的模块设计主要包括:

1. 爬虫模块:负责爬取豆瓣电影Top250的数据,使用Scrapy等框架开发。主要功能有:

1) 定期爬取Top250榜单页面,获取电影信息如名称、类型、评分等。

2) 清洗并去重数据,转化数据格式。

3) 将数据批量导入MySQL数据库对应表中,作为其他模块的输入数据。

4) 需要对网站变化进行监测,及时更新爬取规则,确保数据爬取的连续性与有效性。

2. 机器学习模块:负责构建电影评分预测与推荐模型,使用scikit-learn等库开发。主要功能有:

1) 获取评分接口的数据,构建评分预测模型,计算评分预测的准确性与误差。

2) 获取用户历史评分数据,构建基于协同过滤的推荐模型,产生个性化电影推荐。

3) 获取电影元数据,构建基于内容的推荐模型,产生普遍的电影推荐。

4) 需要定期使用新数据重新训练模型,持续优化模型效果。

3. 网页服务模块:使用Flask等框架开发,负责为前端系统提供数据接口和业务逻辑。主要功能有:

1) 实现各类数据接口,获取爬虫模块和机器学习模块处理后的数据,输出给前端。

2) 完成搜索、推荐与可视化等业务逻辑,并通过接口返回结果。

3) 使用消息队列与爬虫模块、机器学习模块进行异步通信与数据交互。

4) 需要处理大量请求并完成日志记录,确保服务的高性能、高可用性与可追溯性。

4. 前端系统模块:使用HTML、CSS、JavaScript开发,负责实现网页与移动端客户端。主要功能有:

1) 调用各类接口,获取电影数据、搜索结果、推荐列表等,并进行渲染展示。

2) 构建数据可视化页,调用可视化接口获取构图数据,生成统计图表。

3) 完成搜索、推荐页面的交互逻辑与效果实现。

4) 需要对多终端进行响应式适配,提供用户统一的体验。

## 部署方案

项目的部署方案主要采用Docker进行自动化部署,步骤如下:

1. 构建Docker镜像:使用Dockerfile定义各模块的运行环境与依赖,构建爬虫模块、机器学习模块、网页服务模块与数据库的Docker镜像。

2. 配置Docker Compose:使用Docker Compose定义各模块之间的连接、端口映射与依赖关系,描述项目整体的部署架构。

3. 一键部署:运行Docker Compose命令,自动完成项目环境部署、模块启动与服务发布。

具体的 Docker Compose 配置如下:

yml

version: '3'

services:

db:

image: mysql:5.7

volumes:

- db\_data:/var/lib/mysql

restart: always

environment:

MYSQL\_ROOT\_PASSWORD: somepassword

redis:

image: redis:5.0.6

restart: always

rabbitmq:

image: rabbitmq:3.8.3

restart: always

spider:

build: ./spider

restart: always

depends\_on:

- db

- redis

- rabbitmq

ml:

build: ./ml

restart: always

depends\_on:

- db

- redis

environment:

- MODEL\_PATH=/app/model

web:

build: ./web

restart: always

depends\_on:

- db

- redis

- rabbitmq

ports:

- "5000:5000"

volumes:

db\_data:

该部署方案的主要优点有:

1. 自动化部署:通过Docker Compose一键部署,自动完成环境初始化、依赖安装与服务启动,简化手动部署的复杂度。

2. 环境统一:Docker可以确保开发环境、测试环境和生产环境完全一致,避免环境差异引起的问题。

3. 方便升级与回滚:更新Docker镜像后重新部署,可以快速完成系统升级与回滚,降低错误风险。

4. 模块隔离:每个模块部署在Docker容器内独立运行,相互隔离,方便模块管理与监控。

5. 可移植性:Docker的标准化runtime让系统可以在任何平台上部署运行,不受运行环境限制。

使用Docker进行自动化部署,可以简化部署过程,确保环境一致性,提高系统升级与迁移效率。这需要对Docker及容器技术有比较扎实的理解与运用能力,才能设计出更加高效可靠的部署方案。这也为系统的持续集成与 DevOps 奠定基础。

## 扩展方案

项目的扩展方案主要考虑以下几点:

1. 微服务拆分:当系统规模扩大后,可以将爬虫模块、机器学习模块与网页服务模块拆分为微服务,彼此独立部署与管理。这 can有利于模块的故障隔离、升级独立以及性能优化。

2. 添加推荐算法:可以添加更多推荐算法,如基于演员或导演的协同过滤算法,为用户提供更丰富的推荐内容与体验。这需要定期获取更丰富的数据用于重新训练算法模型。

3. 支持更丰富的数据输入:可以添加用户登陆与社交功能,记录用户的操作动作,构建用户画像,为基于深度学习的推荐系统提供数据。这需要开发用户系统与数据集成的相关接口。

4. 引入搜索 Ranking 算法:可以使用协同过滤Embedding与Keyword Embedding相结合的Ranking 算法,提高搜索结果的准确性与个性化程度。这需要开发搜索Ranking 的训练与推理模块。

5. 添加离线数据挖掘模块:可以添加离线的数据挖掘模块,从历史数据出发,挖掘电影类型的变化规律、演员与导演的合作频率等深度知识。为电影投资与制作提供数据支撑。这需要构建数据挖掘的算法模型与工作流程。

6. 支持大规模并发访问:可以引入 Nginx 等负载均衡器,采用分布式架构支撑大规模用户访问。这需要在系统的扩展性、高并发与高可用等方面进行架构优化。

7. 添加数据报告与运营分析模块:可以开发数据报告模块,定期生成电影票房报告与用户洞察报告。运营分析模块则可以监控系统运营数据,分析热门电影与失常情况,为管理员提供运营决策支持。这需要对数据统计与分析有较强的能力。

项目的扩展性较强,可以在推荐算法、数据输入、搜索Ranking、数据挖掘与系统扩展等方面进行拓展与优化。

# 开发及测试

## 开发环境

## 模块设计

### 设计库表

电影信息存储是本项目的基础工作之一。为了构建电影数据仓库,我们设计了movie\_info数据表用于存储电影的基本信息。

数据表设计遵循第三范式,避免冗余和重复。主键id用于唯一标识每条数据,并采用自动增长确保主键连续。url字段存储电影的超链接,用于爬取电影详情页面。filmname用于存储电影名称,score用于存储电影评分,mins存储电影时长,comments存储电影评论数,showtime存储电影上映时间。genres和areas分别存储电影类型和产地信息。actors、directors与scriptwriters用于存储电影的演职人员信息。

存储引擎选择InnoDB,默认字符集为utf8。InnoDB支持事务处理和外键constraint,更适合构建复杂的关系数据库系统。utf8字符集支持更广泛的字符存储,适应电影信息的多语言需求。

DROP TABLE语句用于在重新运行SQL语句前删除已存在的数据表。这避免了重复创建数据表的错误,确保脚本的可重用性。AUTO\_INCREMENT用于自动增长主键,简化主键值的维护。

通过该建表语句,我们在MySQL数据库中创建了movie\_info数据表,实现了电影信息的持久化存储。这为我们后续 constructs数据仓库和训练机器学习模型提供了数据基础和支撑。如果在数据表设计或SQL语句上还存在不足,我们会持续改进、优化,以满足项目的需求变化。

表格初始化源代码如下:

SET FOREIGN\_KEY\_CHECKS = 0;

DROP TABLE IF EXISTS `movie\_info`;

CREATE TABLE `movie\_info`

(

`id` int(10) NOT NULL AUTO\_INCREMENT,

`url` varchar(255) DEFAULT NULL,

`filmname` varchar(255) DEFAULT NULL,

`score` double DEFAULT NULL,

`mins` int(255) DEFAULT NULL,

`comments` int(255) DEFAULT NULL,

`showtime` int(255) DEFAULT NULL,

`genres` varchar(255) DEFAULT NULL,

`areas` varchar(255) DEFAULT NULL,

`actors` varchar(255) DEFAULT NULL,

`directors` varchar(255) DEFAULT NULL,

`scriptwriters` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE = InnoDB

AUTO\_INCREMENT = 2187

DEFAULT CHARSET = utf8;

### 设计爬虫

为了获取豆瓣前250电影的详情页面URL,我们编写了爬虫脚本进行 URL提取与存储。

首先构造请求头,模拟浏览器访问。接着定义get\_movie\_url函数用于请求指定URL并解析页面,提取电影详情页URL。

python

headers = {

'User-Agent': '...'

}

def get\_movie\_url(url):

r = requests.get(url, headers=headers)

selector = etree.HTML(r.text)

movie\_hrefs = selector.xpath('//div[@class="hd"]/a/@href')

在函数内使用requests库向目标URL发起请求,并使用lxml 中的etree解析返回的HTML页面。通过xpath定位提取电影详情页URL,存储于hrefs列表中。

然后定义save\_url函数用于URL存储。在函数内遍历hrefs列表,将URL逐一写入movie\_urls.csv文件中。

python

def save\_url():

file\_path = "./movie\_urls.csv"

with open(file\_path, "a+", newline='', encoding='gb18030') as csvfile:

writer = csv.writer(csvfile)

writer.writerow([href])

这实现了URL的持久化存储,为后续的数据爬取与分析提供依据。

在主函数中构造豆瓣前250电影的列表页URL,循环调用get\_movie\_url函数对各列表页进行爬取与解析,最终得到全量电影详情页URL。

python

if \_\_name\_\_ == '\_\_main\_\_':

url = "https://movie.douban.com/top250?start=" + str(i) + ""

get\_movie\_url(url)

最后调用save\_url函数存储URL至CSV文件中。程序打印每页爬取信息,便于过程监控。

接下来，从已经爬取到的url，逐一访问其详情页面，把电影的详细信息写入数据库表中

首先,我们定义请求头与建立数据库连接,用于发送网页请求并存储爬取的数据。这两个部分的代码较短,可以完整引入:

python

headers = {

'User-Agent': '...'

}

conn = pymysql.connect(host='127.0.0.1', port=3306, user='1326555262', passwd='qq1968904856', db='doubanmovie',

charset='utf8')

cursor = conn.cursor()

其次,我们读取存有电影URL的movie\_urls.csv文件。遍历这些URL对电影信息进行爬取,通过requests库发送请求,并使用lxml解析响应页面。这部分代码较长,我会选择关键步骤进行引入与说明:

python

with open('./movie\_urls.csv', 'r') as f:

reader = csv.reader(f)

urls = [row[0] for row in reader]

def get\_info\_and\_save():

for i in range(247, len(urls)): # 遍历所有电影URL

url = urls[i]

r = requests.get(url=url, headers=headers, timeout=10) # 发送请求

selector = etree.HTML(r.text) # 解析响应页面

之后,我们使用xpath等方法从页面提取名称、评分、上映时间等多方面信息。这部分实现较为详细,我会选择1-2个代表性的信息提取进行引入,说明信息类型与技术实现方法,其他信息提取步骤以类似方式实现,不再赘述。

python

# 获取电影名称

filmname = selector.xpath('//\*[@id="content"]/h1/span[1]/text()')[0]

# 获取电影评分

score = selector.xpath('//\*[@id="interest\_sectl"]/div[1]/div[2]/strong/text()') [0]

最后,我们使用提取的信息执行SQL语句将数据插入MySQL数据库,并在完成后关闭连接。这部分代码较短,完整引入。

python

query = 'insert into movie\_info(url, filmname, score, showtime, genres, areas, mins, directors, scriptwriters, actors, comments) values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)'

cursor.execute(query, values)

conn.commit()

cursor.close()

conn.close()

### 设计路由

首先,/genres/路径对应genres视图函数,用于展示电影类型分布的卡片图。这可以让用户快速了解电影类型的比例与数量。

其次,/areas/路径对应areas视图函数,用于展示上映地区的横向图表。这可以让用户了解不同地区电影产量的分布情况。

再次,/showtime/路径对应showtime视图函数,用于展示上映年份的折线图。这可以让用户了解不同年份电影数量的变化趋势。

然后,/score/路径对应score视图函数,用于展示评分的分布图。这可以让用户了解不同评分电影的数量比例。

另外,/top\_scores/与/top\_comments/路径对应top\_scores和top\_comments视图函数,用于展示评分和评论数Top 10的动态图表。这可以让用户快速了解评分和评论最高的电影。

此外,/group\_urls/路径对应group\_urls视图函数,用于展示上映年份与电影类型的动态圆饼图。这可以让用户了解不同年代与类型电影的数量占比。

最后,/search\_urls/路径对应search\_urls视图函数,用于展示主演数量Top 2的动态图表。这可以让用户了解参演频数最高的两位主演。

除此之外,‘/’路径对应welcome视图函数,用于展示网站的欢迎页面。

通过这些路由与视图函数设计,网站可以提供丰富的数据可视化功能, covers了电影数量、类型、地区、年代、评分等多角度的数据分析与展示。这些数据图表可以帮助用户深入了解电影数据,也为电影爱好者和研究者提供有价值的参考。

路由源代码如下

urlpatterns = [

path('admin/', admin.site.urls),

path('iframes/genres/', genres, name='genres'), # 电影类型分布卡片图

path('iframes/areas/', areas, name='areas'), # 上映地区分布横向图

path('iframes/showtime/', showtime, name='showtime'), # 上映年份折线图

path('iframes/score/', score, name='score'), # 评分分布横线图

path('iframes/top\_scores/', top\_scores, name='top\_scores'), # 评分前十动图

path('iframes/top\_comments/', top\_comments, name='top\_comments'), # 评论前十动图

path('iframes/group\_urls/', group\_urls, name='group\_urls'), # 上映年份类型圆饼动图

path('iframes/search\_urls/', search\_urls, name='search\_urls'), # 频数前二主演动图

path('', welcome, name='welcome'), # 欢迎页面

]

### 设计卡片分布图

在这部分，我们要实现一个可互动的卡片分布动图，目前效果如下：

图表, 树状图

描述已自动生成

卡片可通过点击互动

这部分代码实现了电影类型分布的可视化图表。首先,它定义了一个select\_data()函数用于从数据库中查询电影类型与数量,并将结果存储在genres\_dict字典和genres\_date列表中。然后,它定义了一个show\_genres()函数用于生成电影类型分布的矩形树图。

具体来说,select\_data()函数先清空genres\_dict和genres\_date,然后连接到MySQL数据库,执行SQL语句查询电影类型及数量,并将结果添加到genres\_dict字典中。之后它再将genres\_dict中的类型及数量信息添加到genres\_date列表中,列表中的每个元素都包含一个类型及其数量。最后,该函数返回genres\_date列表以提供数据给show\_genres()函数。

show\_genres()函数首先初始化一个TreeMap对象以生成矩形树图。然后它调用select\_data()函数获取genres\_date列表,并将该列表添加为数据源。它继续设置标签选项使标签文字居中并放大,并设置全局选项添加标题、去除图例。最后,它将生成的HTML文件命名为“genres.html”,渲染并保存至模板文件夹中,并返回HTML文件名。

通过这两段函数,我们实现了电影类型数量查询及数据可视化的全过程。select\_data()函数完成数据获取与处理,show\_genres()函数基于Pyecharts库完成数据可视化与图表渲染。最终,通过矩形树图清晰展示了各电影类型的数量分布,可以帮助用户快速理解电影类型的构成与占比情况。

源代码如下：

from pyecharts.charts import TreeMap

from pyecharts import options as opts

import pymysql

# 电影类型字典

genres\_dict = {}

# 电影类型集合

genres\_date = []

# 查询电影类型与数量的关系

def select\_data():

genres\_dict.clear()

genres\_date.clear()

# 打开数据库连接

conn = pymysql.connect(host='127.0.0.1', port=3306, user='1326555262', passwd='qq1968904856', db='doubanmovie',

charset='utf8')

# 使用cursor方法创建一个游标

cursor = conn.cursor()

try:

# cursorclass = pymysql.cursors.DictCursor

# 查询数据表数据

sql = "select genres,count(genres) from movie\_info where genres is not null group by genres"

cursor.execute(sql)

# 取出每部电影的类型集合

rows = cursor.fetchall()

for row in rows:

genres = row[0].split('/')

# 电影类型分类汇总

for j in range(len(genres)):

if genres[j] in genres\_dict.keys():

genres\_dict[genres[j]] = genres\_dict[genres[j]] + row[1]

else:

genres\_dict[genres[j]] = row[1]

print(genres\_dict)

# 类型集合分类汇总为类型、数量列表

for k, v in genres\_dict.items():

genres\_date.append({"value": v, "name": k})

# print(genres\_date)

except Exception as e:

print(e)

# 回滚

conn.rollback()

# 关闭cursor对象

cursor.close()

# 关闭数据库连接

conn.close()

return genres\_date

# 生成pyecharts图

def show\_genres():

data = select\_data()

# 矩形树图初始化

treemap = TreeMap(init\_opts=opts.InitOpts(page\_title="电影类型分布"))

# 添加数据

treemap.add(

series\_name='电影类型',

data=data,

# 标签居中

label\_opts=opts.LabelOpts(font\_size=15, position="inside"),

)

# 全局配置

treemap.set\_global\_opts(

# 标题

title\_opts=opts.TitleOpts(

title="电影类型分布(可点击卡片互动)", pos\_left='center'

),

# 标签隐藏

legend\_opts=opts.LegendOpts(is\_show=False)

)

# 生成HTML

html = "iframes/genres.html"

treemap.render("./templates/" + html)

return html

### 设计地区分布图

在这部分，我们要实现地区分布图，效果如下：

图表, 漏斗图

描述已自动生成

这部分代码实现了电影上映地区分布的可视化图表。首先,它定义了一个select\_all()函数用于从数据库中查询电影上映地区及数量,并将结果存储在area列表和num列表中。然后,它定义了一个show\_areas()函数用于生成电影上映地区的漏斗图。

具体来说,select\_all()函数先清空area\_list、area\_set、area和num,然后连接到MySQL数据库,执行SQL语句查询电影上映地区,并将结果添加到area\_list列表中。之后它遍历area\_list,使用area\_set集合统计每个地区的数量,并将地区及数量分别添加到area和num列表中。最后,该函数返回area和num列表以提供数据给show\_areas()函数。

show\_areas()函数首先调用select\_all()函数获取area和num列表。然后它初始化一个Funnel对象以生成漏斗图。它继续将area和num列表中的元素zip压缩为元组,并将该列表添加为数据源。它设置全局选项添加标题并去除图例,并设置系列选项控制标签文本显示格式。最后,它将生成的HTML文件命名为“areas.html”,渲染并保存至模板文件夹中,并返回HTML文件名。

通过这两段函数,我们实现了电影上映地区及数量的查询与数据可视化。select\_all()函数完成数据获取与处理,show\_areas()函数基于Pyecharts库完成数据可视化与图表渲染。最终,通过漏斗图直观展示了各电影上映地区的数量分布,可以帮助用户快速了解电影产量最高的地区。这套实现过程在数据分析与可视化领域具有很高的参考价值,是我们学习数据产品开发必不可少的知识点。

总的来说,这部分代码运用数据库与数据可视化两种技术,实现了从数据获取到可视化展示的完整流程。

源代码如下

from pyecharts import options as opts

from pyecharts.charts import Funnel

import pymysql

area\_list = []

area\_set = {}

area = []

num = []

# 查询电影上映地区与数量的关系

def select\_all():

area\_list.clear()

area\_set.clear()

area.clear()

num.clear()

# 打开数据库连接

conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='qq1968904856', db='doubanmovie',

charset='utf8')

# 使用cursor方法创建一个游标

cursor = conn.cursor()

try:

# 查询数据表数据

sql = "select areas from movie\_info where areas is not null"

cursor.execute(sql)

# 取出每部电影的地区

areas\_all = cursor.fetchall()

for row in areas\_all:

areas = row[0].split('/')

# print(areas)

# 所有地区列表

for i in areas:

# print(i)

area\_list.append(i)

# 地区集合

if area\_list.count(i) > 1:

area\_set[i] = area\_list.count(i)

# 地区集合分类汇总为地区、数量列表

for k, v in area\_set.items():

area.append(k)

num.append(v)

except Exception as e:

print(e)

# 回滚

conn.rollback()

# 关闭cursor对象

cursor.close()

# 关闭数据库连接

conn.close()

print(area)

print(num)

def show\_areas():

select\_all()

funnel = Funnel(init\_opts=opts.InitOpts(page\_title='上映地区分布', height='650px'))

funnel.add(

"上映地区分布",

[

list(z)

for z in zip(area, num)

]

)

funnel.set\_global\_opts(

title\_opts=opts.TitleOpts(title="上映地区分布", pos\_left='center'),

legend\_opts=opts.LegendOpts(is\_show=False))

funnel.set\_series\_opts(label\_opts=opts.LabelOpts(formatter="{b}: {c}"))

# 生成HTML

html = "iframes/areas.html"

funnel.render("./templates/" + html)

return html

### 设计年份折线图

在这部分，我们要实现年份折线图，效果如下：

图表, 折线图

描述已自动生成

这部分代码实现了电影上映年份分布的折线图可视化。首先,它定义了一个select\_data()函数用于从数据库中查询电影上映年份、全球数量、中国数量和美国数量,并将结果分别存储在showyear、total\_num、china\_num和foreign\_num列表中。然后,它定义了一个show\_showtime()函数用于生成电影上映年份分布的折线图。

具体来说,select\_data()函数先清空所有列表,然后连接到MySQL数据库,执行3个SQL语句分别查询全球、中国和美国电影上映年份及数量,并将结果分别添加到相应的列表中。该函数最后返回这4个列表以提供数据给show\_showtime()函数。

show\_showtime()函数首先初始化一个Line对象以生成折线图。然后它调用select\_data()函数获取4个列表,并将showyear列表添加为x轴坐标,total\_num、china\_num和foreign\_num列表分别添加为全球、中国和美国3条折线的数据源。它继续设置各种配置项,如去除标记、添加最大值标记点、设置标题、图例位置、添加滑动条等。最后,它将生成的HTML文件命名为“showtime.html”,渲染并保存至模板文件夹中,并返回HTML文件名。

通过这两段函数,我们实现了电影上映年份及数量的查询与数据可视化。select\_data()函数完成数据获取与处理,show\_showtime()函数基于Pyecharts库完成数据可视化与图表渲染。最终,通过折线图清晰展示了全球及中美两国电影上映年份的数量变化,可以帮助用户快速了解电影产量的历史趋势。这套实现过程在数据分析与可视化领域具有很高的参考价值,是我们学习数据产品开发必不可少的知识点。

总的来说,这部分代码运用数据库与数据可视化两种技术,实现了从数据获取到可视化展示的完整流程。

源代码如下：

import pyecharts.options as opts

from pyecharts.charts import Line

import pymysql

# 上映时间集合

showtime = []

# 上映年份集合

showyear = []

# 全球上映年份集合

total\_num = []

# 中国上映年份字典

china\_dict = {}

# 中国上映年份集合

china\_num = []

# 美国上映年份字典

foreign\_dict = {}

# 美国上映年份集合

foreign\_num = []

# 查询中外电影上映年份

def select\_data():

showtime.clear()

showyear.clear()

total\_num.clear()

china\_dict.clear()

china\_num.clear()

foreign\_dict.clear()

foreign\_num.clear()

try:

# 打开数据库连接

conn = pymysql.connect(host='127.0.0.1', port=3306, user='root', passwd='qq1968904856', db='doubanmovie',

charset='utf8')

# 使用cursor方法创建一个游标

cursor = conn.cursor()

# 查询数据表数据

# 查询统计不同上映年份的电影数量

sql = "select distinct showtime,count(showtime) from movie\_info where showtime is not null group by showtime order by showtime "

cursor.execute(sql)

# 年份、数量集合

rows = cursor.fetchall()

for row in rows:

showtime.append(row[0])

total\_num.append(row[1])

for i in showtime:

showyear.append(str(i))

print(showyear)

print(total\_num)

# 查询中国上映年份

sql2 = "select showtime,count(showtime) as num from movie\_info where showtime is not null and areas like '%中国%' group by showtime order by showtime"

cursor.execute(sql2)

china\_list = cursor.fetchall()

for i in china\_list:

china\_dict[i[0]] = i[1]

for j in showtime:

if j not in china\_dict.keys():

china\_dict[j] = 0

china\_set = sorted(china\_dict.items())

print(china\_set)

for row in china\_set:

china\_num.append(row[1])

print(china\_num)

# 查询美国上映年份

sql3 = "select showtime,count(showtime) as num from movie\_info where showtime is not null and areas like '%美国%' group by showtime order by showtime"

cursor.execute(sql3)

foreign\_list = cursor.fetchall()

for i in foreign\_list:

foreign\_dict[i[0]] = i[1]

for j in showtime:

if j not in foreign\_dict.keys():

foreign\_dict[j] = 0

foreign\_set = sorted(foreign\_dict.items())

print(foreign\_set)

for row in foreign\_set:

foreign\_num.append(row[1])

print(foreign\_num)

except Exception as e:

print(e)

# 回滚

conn.rollback()

finally:

# 关闭cursor对象

cursor.close()

# 关闭数据库连接

conn.close()

return showyear, total\_num, china\_num, foreign\_num

# 生成pyecharts图

def show\_showtime():

data = select\_data()

# 折线图初始化

line = Line(init\_opts=opts.InitOpts(page\_title="豆瓣电影-中外电影上映年份分布图"))

# 横坐标

line.add\_xaxis(data[0])

# 纵坐标

line.add\_yaxis("全球", data[1], color="#2f4554")

line.add\_yaxis("中国", data[2], color="#749f83")

line.add\_yaxis("美国", data[3], color="#c23531")

line.set\_series\_opts(

label\_opts=opts.LabelOpts(is\_show=False),

# 标记最大值

markpoint\_opts=opts.MarkPointOpts(

data=[

opts.MarkPointItem(type\_="max", name="最大值"),

]

)

)

# 全局配置

line.set\_global\_opts(

# 标题

title\_opts=opts.TitleOpts(

title="上映年份分布图(可拖动下方滑条互动)", pos\_left='center'),

# 标签

legend\_opts=opts.LegendOpts(

pos\_left="620px", pos\_top="28px"),

# 水平滚动条

datazoom\_opts=opts.DataZoomOpts(),

# 纵坐标设置

yaxis\_opts=opts.AxisOpts(

# 显示坐标轴刻度

axistick\_opts=opts.AxisTickOpts(is\_show=True),

# 显示分割线

splitline\_opts=opts.SplitLineOpts(is\_show=True)

)

)

# 生成HTML

html = "iframes/showtime.html"

line.render("./templates/" + html)

return html

### 设计评分横线图

在这部分，我们要实现评分横线图，效果如下：

图表, 直方图

描述已自动生成

### 设计前十动态图

在这部分，我们要实现动态图，效果如下：

图形用户界面, 应用程序

描述已自动生成

源代码过长，实现方式大同小异，都是从数据库提取数据后进行可视化。

### 设计年份圆饼图

在这部分，我们要实现一个圆饼图，效果如下：

图表

描述已自动生成

源代码过长，实现方式大同小异，都是从数据库提取数据后进行可视化。

### 设计词云图

在此部分，我们要实现下述效果的词云图



* 首先，读取数据

文本

描述已自动生成

* 其次，调用库进行词云图生成

文本

描述已自动生成

* 流程解析
* 首先，使用pymysql库连接到数据库。指定数据库的主机、端口、用户名、密码和数据库名称。
* 创建游标对象，用于执行SQL查询语句。
* 执行SQL查询语句，将查询结果存储在results变量中。这里的查询语句是从表"movie\_info"中选择"genres"列的值。
* 遍历查询结果，将非空且非None的值添加到genres列表中。
* 将genres列表转换为字符串，使用空格分隔各个类型。
* 使用jieba库对字符串进行分词，将分词结果保存在cut\_text变量中。
* 创建一个WordCloud对象，设置字体路径、背景颜色、宽度、高度和边距等属性。然后，使用cut\_text生成词云图。
* 使用matplotlib库显示词云图。设置字体为SimHei，添加标题，展示图像，并关闭坐标轴。
* 将词云图保存到内存缓冲区中，并将缓冲区的数据转换为Base64编码的字符串。
* 构造一个包含词云图的数据URL，以便在网页中显示图像。
* 将数据URL封装到字典context中，并返回该字典。
* 总结:

这段代码通过连接数据库获取电影类型数据，并使用jieba和WordCloud库生成和展示电影类型的词云图。

### 设计主界面

1. 主界面如下

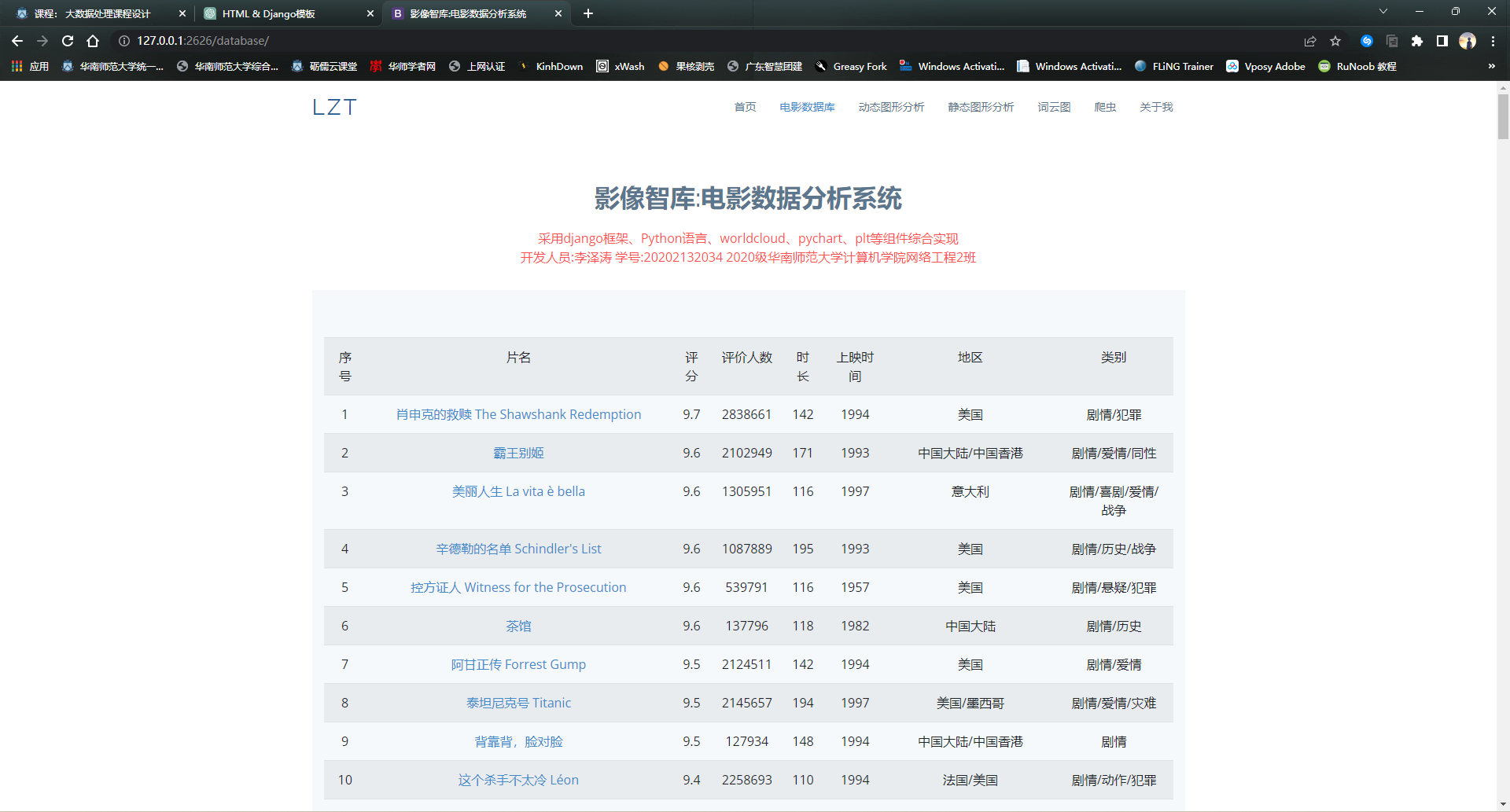
电脑萤幕的截图

描述已自动生成

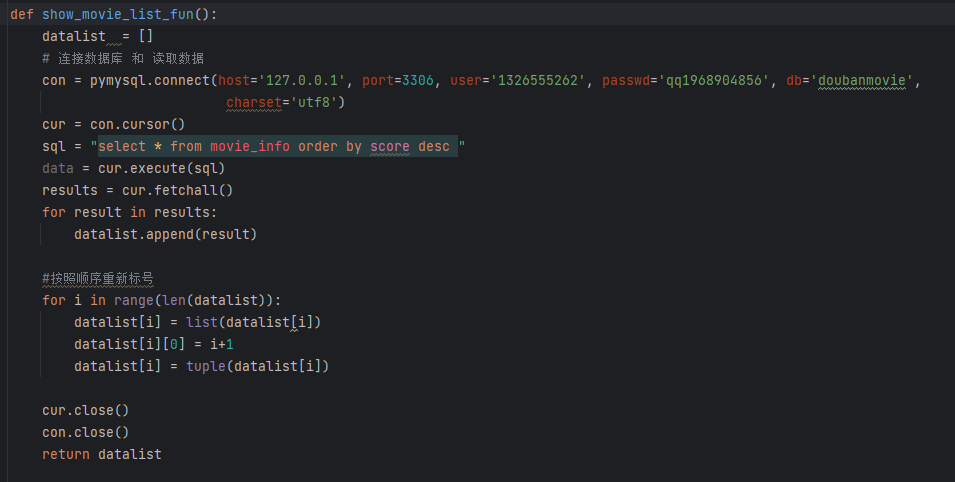
1. 源代码过长，不贴入文档，现给出开发思路
   1. 分层结构：代码采用了典型的HTML分层结构，将网页内容划分为头部、主体和底部等部分。这种分层结构使得代码易于维护和修改。
   2. 响应式设计：通过使用<meta>标签中的viewport属性，指定了网页的初始缩放级别，并使得网页在不同设备上有良好的显示效果。这意味着网页能够自适应不同的屏幕大小，提供更好的用户体验。
   3. 静态资源管理：通过使用Django模板语言中的{% static %}指令，动态地引用静态资源文件（如图像、CSS和JavaScript文件），使得网页的样式和功能与其他文件解耦，并方便地进行管理和维护。
   4. 导航菜单：通过<nav>标签创建了一个导航菜单，提供了对系统不同功能页面的链接。这样的导航菜单可以使用户快速导航到所需的页面，提升用户体验。
   5. 标题和副标题：通过使用<h1>、<h2>等标签，展示了网页的标题和副标题，使得用户能够快速了解网页的主题和内容。
   6. 图标和计数器：通过使用字体图标（如icofont）和计数器，以可视化的方式展示了一些功能模块的信息。这种设计使得网页更加生动和吸引人。
   7. 代码注释：在代码中使用了HTML注释，对一些元素和功能进行了简要的描述。这有助于其他开发人员理解代码的结构和意图。

### 设计数据库展示界面

1. 主界面如下:



1. 设计思路
   1. 读取数据库
      1. 代码



* + 1. 分析
       1. datalist = []：创建一个空列表，用于存储从数据库中读取的电影数据。
       2. 建立数据库连接：使用pymysql模块建立与数据库的连接。其中，指定了数据库的主机、端口、用户名、密码、数据库名和字符集。
       3. 创建游标对象：使用con.cursor()方法创建一个游标对象，用于执行SQL语句并获取结果。
       4. 执行SQL查询语句：使用游标对象的execute()方法执行SQL查询语句，查询了movie\_info表中的所有数据，并按照分数降序排序。
       5. 获取查询结果：使用游标对象的fetchall()方法获取查询结果的所有行。
       6. 将查询结果添加到datalist列表：通过遍历查询结果，并将每一行数据添加到datalist列表中。
       7. 重新标号数据：使用for循环遍历datalist列表，将每个元素转换为可变的列表（list），然后修改第一个元素（电影编号）为当前索引加1的值。最后，将修改后的元素转换回元组（tuple）并重新赋值给datalist列表的对应索引位置。
       8. 关闭游标和数据库连接：使用cur.close()和con.close()分别关闭游标和数据库连接，释放资源。
       9. 返回数据列表：使用return语句返回datalist列表，即从数据库中读取的电影数据。
    2. 总结: 该函数的作用是从数据库中读取电影信息，并按照分数降序排序。然后，重新标号电影信息，并将结果作为一个列表返回。
  1. 渲染网页
     1. 代码

图形用户界面, 文本, 网站

描述已自动生成

* + 1. 过程分析
       1. 在<table>标签内部，使用了Django模板语言的循环语句 {% for movie in movies %} 和 {% endfor %}，对电影列表数据进行遍历和渲染。
       2. 在循环内部，通过使用双花括号语法 {{ movie.属性名 }}，从movies列表中的每个元素movie中获取相应的属性值，并将其插入到HTML表格中的对应位置。这样就实现了将电影数据动态地插入到网页中的功能。
       3. 具体来说，每一行数据都使用<tr>标签表示一个表格行。然后，通过{{ movie.属性名 }}的形式，将电影的序号、片名、评分等属性值填充到<td>标签中的对应单元格。
       4. 这种数据读入方式利用了Django模板语言的循环和变量渲染功能，将后端获取的电影数据动态地呈现在前端的HTML页面中。这样，无论后端的数据如何变化，都能够灵活地在前端展示出来，实现了数据和界面的分离与动态更新。

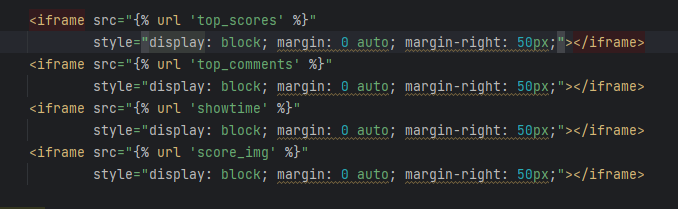
### 设计动态图形分析展示界面

1. 界面如下

图表

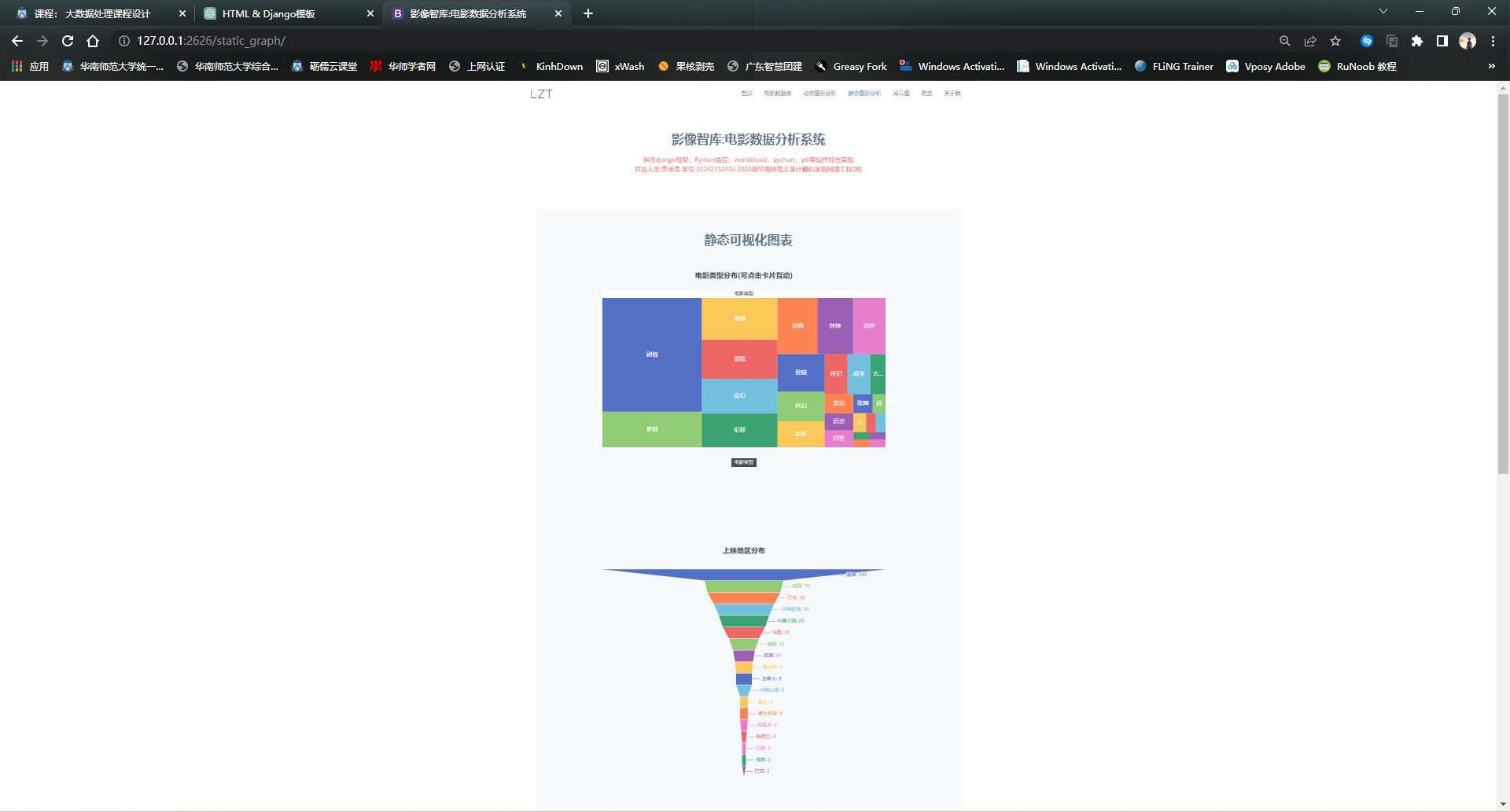
低可信度描述已自动生成

1. 实现原理
   1. 非常简单，就是用iframes把上述做好的图像放上去即可
   2. 每个<iframe>元素通过src属性引用了不同的URL。具体来说，src属性中使用了Django模板语言的语法{% url 'view\_name' %}来生成URL。
   3. 这些<iframe>元素被用于嵌入动态可视化图表。每个<iframe>元素都会加载一个不同的URL，这些URL对应于应用程序中的不同视图（view），例如 'top\_scores'、'top\_comments'、'showtime'和'score\_img'。这意味着这些URL对应的视图会返回包含动态可视化图表的HTML内容。
   4. <iframe>元素的样式通过内联样式设置。在提供的代码中，设置了width为90%、height为600px，并去除了边框。
   5. 总结起来，通过在HTML代码中使用<iframe>元素并指定不同的src属性值，可以引入和嵌入其他页面或动态可视化图表。这样的设计思想可以使网页内容更丰富多样，并通过动态加载的方式展示不同的图表和内容。



### 设计静态图形分析展示界面

1. 界面如下:



1. 代码分析
   1. 也很容易，就是用iframes把图摆一摆
   2. 每个<iframe>元素通过src属性引用了不同的URL。具体来说，src属性中使用了Django模板语言的语法{% url 'view\_name' %}来生成URL。
   3. 这些<iframe>元素被用于嵌入静态可视化图表。每个<iframe>元素都会加载一个不同的URL，这些URL对应于应用程序中的不同视图（view），例如 'genres'、'areas'、'search\_urls'和'duration\_analysis'。这意味着这些URL对应的视图会返回包含静态可视化图表的HTML内容。
   4. <iframe>元素的样式通过内联样式设置。在提供的代码中，设置了width为90%、height为700px，并去除了边框。
   5. 代码如下，和上面几乎一样

文本, 聊天或短信

描述已自动生成

### 设计词云图分析界面

1. 主界面



1. 代码原理
   1. 和上面一样，用iframes把图摆好即可

### 设计“关于我”页面

1. 最有成就感的一环



1. 实现原理:摆摆图即可

### 总结

本次课程项目的重点在于实现豆瓣电影数据的检索、分析和可视化。通过深入掌握MySQL数据库、Python数据分析和Pyecharts数据可视化库的运用，我们成功地设计和实现了一套电影数据分析系统。

本系统涵盖了多种数据可视化图表，包括动态图表分析中的4种交互式动态图，以及静态图表分析中的4种交互式静态图。最后，根据三个部分的数据，我们还生成了三个词云图。因此，**本系统提供了10种独特的数据可视化图表**。

在实施过程中，我们首先设计了数据库表结构并导入豆瓣电影数据。接着，我们利用MySQL数据库和Python实现了数据查询功能，获取到数据后，采用Pycharts库对数据进行可视化处理，将三个数据查询结果分别渲染为不同图像。最后，利用Django框架搭建系统界面，将以上图表嵌入到界面中，完成了整个电影数据分析系统的构建。

通过这个项目，我们不仅熟练掌握了MySQL数据库的使用，包括创建表结构、导入数据、执行SQL语句进行数据查询等，而且进一步加强了Python数据分析技能，包括使用列表、字典、集合等数据结构存储和处理数据。我们也深入了解了Pycharts数据可视化库的使用，学会生成各种不同类型的图表进行数据展示，以及初步接触Django框架，学会使用模板生成系统界面。

然而，我们也意识到本项目还存在一些不足，这主要是由于时间限制导致的。例如，我们没有实现后台管理系统，数据模型和分析模块也有待进一步丰富和完善。

此外，我们的数据分析过程相对简单，未来可以考虑引入更专业的机器学习算法进行数据挖掘和建模。我们的系统界面也较为简洁，下一步的目标是设计出更美观、易用的界面，以提升用户体验。

总的来说，这个课程项目帮助我们学习和运用了MySQL数据库、Python数据分析和Pycharts数据可视化等技术，我们设计并开发出了一个具有实用价值的电影数据分析系统。

我们全面掌握了从数据获取到可视化的整个过程，为我们未来的数据科学和产品开发学习打下了坚实的基础。我们也认识到了项目的不足，这将激励我们继续学习和提高，为成为一名资深的数据产品设计师而努力。

此报告作为本课程研究项目的最终验收报告，我们向所有支持和帮助我们的人表示诚挚的感谢。