



数眸 - 智能数据分析平台










项目简介

数眸是一款基于Streamlit构建的现代化数据分析应用，让数据洞察如眸般清澈明亮。该平台集成了数据上传、清洗、分析、可视化和机器学习等完整功能，并配备了AI智能助手，为用户提供专业的数据分析指导。



核心特色

-  **数眸品牌**：让数据洞察如眸般清澈明亮
-  **多模式设计**：新手模式、普通模式、专业模式，满足不同用户需求
-  **AI智能助手**：集成大语言模型，提供智能分析和建议
-  **丰富可视化**：支持20+种图表类型，包括3D图表和交互式图表
-  **专业统计分析**：提供完整的描述性和推断性统计功能
-  **机器学习集成**：支持分类、回归、聚类等多种ML算法
-  **现代化UI**：采用Material Design 3设计系统，界面美观易用



重构概述

本次重构将原始的单一文件应用（`app.py`）重构为模块化架构，实现了**页面绘制功能与实际程序功能的分离**，提高了代码的可维护性、可扩展性和可读性。



重构后的项目结构

DataAnalysis_ver2/

└─ `app.py`

原始应用文件（保留）

```
├─ app_refactored.py          # 重构后的主应用文件

├─ src/                       # 源代码目录
│   ├── __init__.py
│   ├── config/               # 配置模块
│   │   ├── __init__.py
│   │   └─ settings.py        # 应用配置和样式
│   ├── modules/              # 页面模块
│   │   ├── __init__.py
│   │   └─ pages.py           # 页面渲染功能
│   └─ utils/                  # 工具模块
│       ├── __init__.py
│       ├── data_processing.py # 数据处理工具
│       ├── visualization_helpers.py # 可视化工具
│       └─ ml_helpers.py       # 机器学习工具

├─ requirements.txt

└─ requirements_python313.txt
```

重构原则

1. 分离关注点 (Separation of Concerns)

- **UI层:** 页面绘制和用户交互
- **业务逻辑层:** 数据处理、分析和机器学习
- **配置层:** 应用设置和样式管理

2. 模块化设计

- 每个模块负责特定功能
- 模块间通过清晰的接口通信
- 降低模块间耦合度

3. 代码复用

- 提取公共功能到工具模块
- 避免代码重复
- 提高开发效率

模块详细说明

1. 配置模块 (`src/config/settings.py`)

功能: 集中管理应用的所有配置参数

包含内容:

- 页面配置 (`PAGE_CONFIG`)
- 导航页面配置 (`NAV_PAGES`)
- 自定义CSS样式 (`CUSTOM_CSS`)
- 支持的文件格式 (`SUPPORTED_FILE_TYPES`)

- 组件兼容性配置 (COMPONENT_AVAILABILITY)
- 机器学习配置 (ML_CONFIG)
- 可视化配置 (VISUALIZATION_CONFIG)
- 统计分析配置 (STATISTICAL_CONFIG)
- 数据清洗配置 (DATA_CLEANING_CONFIG)
- 作者信息 (AUTHOR_INFO)

优势:

- 配置集中管理，易于维护
- 支持不同环境的配置切换
- 减少硬编码

2. 数据处理工具 (src/utils/data_processing.py)

功能: 提供数据处理的核心功能

主要函数:

- load_data(): 数据加载 (带缓存)
- calculate_data_quality_score(): 数据质量评分
- get_data_info(): 获取数据基本信息
- handle_missing_values(): 处理缺失值
- handle_outliers(): 处理异常值
- handle_duplicates(): 处理重复值
- clean_string_data(): 清洗字符串数据
- convert_data_format(): 数据格式转换

优势:

- 功能独立，易于测试
- 支持缓存，提高性能
- 类型提示，提高代码质量

3. 可视化工具

(`src/utils/visualization_helpers.py`)

功能: 提供各种图表创建功能

主要函数:

- `create_bar_chart()`: 创建柱状图
- `create_line_chart()`: 创建折线图
- `create_scatter_chart()`: 创建散点图
- `create_heatmap()`: 创建热力图
- `create_3d_scatter()`: 创建3D散点图
- `create_radar_chart()`: 创建雷达图
- `create_missing_values_chart()`: 创建缺失值分析图
- `create_feature_importance()`: 创建特征重要性图

优势:

- 统一的图表创建接口
- 支持自定义主题
- 易于扩展新的图表类型

4. 机器学习工具 (`src/utils/ml_helpers.py`)

功能: 提供机器学习相关功能

主要函数:

- `validate_data_for_ml()`: 验证数据是否适合ML任务
- `preprocess_data_for_ml()`: 数据预处理
- `train_classification_model()`: 训练分类模型
- `train_regression_model()`: 训练回归模型
- `perform_clustering()`: 执行聚类分析
- `perform_cross_validation()`: 执行交叉验证
- `generate_learning_curve()`: 生成学习曲线

- `perform_feature_engineering()` : 特征工程

优势:

- 标准化的ML流程
- 支持多种算法
- 完整的评估指标

5. 页面模块 (`src/modules/pages.py`)

功能: 负责页面渲染和UI交互

主要函数:

- `render_home_page()` : 渲染首页
- `render_sidebar()` : 渲染侧边栏
- `render_footer()` : 渲染页脚

优势:

- UI逻辑与业务逻辑分离
- 页面组件可复用
- 易于维护和修改



使用方法

运行重构后的应用

```
# 运行重构后的应用

streamlit run app_refactored.py

# 或运行原始应用（保留）
```

```
streamlit run app.py
```

导入模块使用

```
# 导入配置
```

```
from src.config.settings import PAGE_CONFIG, NAV_PAGES
```

```
# 导入数据处理工具
```

```
from src.utils.data_processing import load_data,  
calculate_data_quality_score
```

```
# 导入可视化工具
```

```
from src.utils.visualization_helpers import  
create_bar_chart, create_heatmap
```

```
# 导入机器学习工具
```

```
from src.utils.ml_helpers import train_classification_model,  
perform_clustering
```



重构效果对比

重构前

- 单一文件，2596行代码
- 功能混合，难以维护
- 代码重复，难以复用
- 配置分散，难以管理

重构后

- 模块化架构，职责清晰
- 功能分离，易于维护
- 代码复用，提高效率
- 配置集中，易于管理

迁移指南

从原始代码迁移到重构代码

1. **保留原始文件:** `app.py` 保持不变，确保向后兼容
2. **使用新架构:** 新功能开发使用重构后的模块化架构
3. **逐步迁移:** 可以逐步将原始代码中的功能迁移到新模块

添加新功能

1. **确定功能类型:** 数据处理、可视化、机器学习等
2. **选择对应模块:** 在相应的工具模块中添加函数
3. **更新配置:** 如需要，在 `settings.py` 中添加配置
4. **集成到页面:** 在 `pages.py` 或主应用中调用新功能

测试建议

单元测试

```
# 测试数据处理函数
```

```
def test_load_data():
```



```
# 测试数据加载功能
```

```
pass
```

```
def test_calculate_data_quality_score():
```

```
# 测试数据质量评分
```

```
pass
```

集成测试

```
# 测试完整的数据分析流程
```

```
def test_data_analysis_pipeline():
```

```
# 测试从数据加载到报告生成的完整流程
```

```
pass
```



性能优化

缓存机制

- 使用 `@st.cache_data` 装饰器缓存数据加载
- 使用 `@st.cache_resource` 装饰器缓存模型

内存优化

- 及时释放不需要的数据

- 使用生成器处理大数据集



未来扩展

可能的扩展方向

1. **插件系统:** 支持第三方插件
2. **API接口:** 提供RESTful API
3. **数据库集成:** 支持数据库连接
4. **云部署:** 支持云端部署
5. **多用户支持:** 支持多用户并发

扩展建议

1. **保持模块化:** 新功能继续遵循模块化设计
2. **接口稳定:** 保持公共接口的稳定性
3. **文档更新:** 及时更新文档和示例
4. **版本控制:** 使用语义化版本控制