



数据处理（统一合并）= 清洗 + 格式

数据提交格式（格式统一）：

- ☒ 智联DE行信息爬取错误问题
- ☒ 输出编码utf-8
- ☒ 表头未添加 数据合并时多增加一列——“来源”
- ☒ 时间统一（格式：xxxx-xx-xx 年-月-日）
- ☒ 地区 北京 昌平区 回龙观 → 北京 昌平
- ☒ 薪资统一（30k 60k 14薪）
 - ☒ “薪资面议”，“面议” → “薪资面议”
 - ☒ 保留“急聘”
 - ☒ 保留“1000-1500元/周” → “xxxx-xxxx元/周”
 - ☒ 保留“120-150元/时” → “xxx-xxx元/时”
 - ☒ 保留“14000-15000元/月” → “14k-15k”
 - ☒ 保留“100-200/天” → “xxx-xxx/天”
- ☒ 1年以内、一年以下 → 1年以下
- ☒ 无经验 → 经验不限，应届生单独保留
- ☒ 统招本科 → 本科
- ☒ 企业规模

boss	猎聘	智联	统一格式
0-20人	1-49人	20人以下	0-20人
20-99人	50-99人	20-99人	保留
100-499人	100-499人	100-299人	保留
500-999人	500-999人	300-499人	保留
1000-9999人	1000-2000人	500-999人	保留
10000人以上	2000-5000人	1000-9999人	保留
（空白）	5000-10000人	（空白）	保留
	10000人以上		保留
	（空白）		保留

- ☒ y 融资情况和规模
- ☐ 合并汇总汇报：
 - ☒ 数据总量
 - ☒ 删除重复项
 - ☐ 核验过程和内容，排查异常值（如：月薪3000k），保留有空白项的数据
- ☐ 线上共享文档
- ☒ 合并不同来源的数据（分享脚本），标识来源（三个网站通过添加字段区分）

数据统一说明文件:

数据清洗过程

```
import pandas as pd

# 删除岗位要求列中的空白项
df = df.dropna(subset=['岗位要求'])

# 把所有空数据替换成“无”
df.fillna("无", inplace=True)

# 对所有列数据strip
df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)
```

数据的格式统一过程

第一步：[yz合并详情页](#)

1. y合并详情页

读取文件

检测到文件的编码格式有gb2312、gb18030，统一用gb2312编码打开文件

```
import pandas as pd
import chardet

def detect_encoding(file_path):
    with open(file_path, 'rb') as f:
        result = chardet.detect(f.read())
        encoding = result['encoding'].lower()
        if encoding == 'gb2312':
            encoding = 'gb18030' # Use 'gb18030' instead of 'gb2312'
    return encoding

# 检测文件编码
file_encoding = detect_encoding('A20231225.csv')
# 输出检测到的编码
print(f"Detected encoding: {file_encoding}")
# 读取CSV文件
df1 = pd.read_csv('A20231225.csv', encoding=file_encoding)
# 删除最后一列
df1 = df1.iloc[:, :-1]

# 检测文件编码
file_encoding = detect_encoding('B20231225.csv')
# 输出检测到的编码
print(f"Detected encoding: {file_encoding}")
# 读取CSV文件
df2 = pd.read_csv('B20231225.csv', encoding=file_encoding)
```

大致页文件示例（10列）

岗位名称	地区/区位	薪资	工作年限	学历	公司名	企业类别	融资情况
------	-------	----	------	----	-----	------	------

删除“详情页链接”列

AI人工智能	北京-东城	17-30k	3-5年	本科	某北京运营运营商/增	50-99人	https://www.liepin.com/a/51549433.shtml?pgRef=c_pc
AI人工智能	北京	30-60k	1-3-5年	统招本科	某国内IT服IT服务	已上市	10000人以
AI人工智能	北京-海淀	15-20k	1-3年	硕士	铭台(北京)电子商务	战略投资	50-99人
AI人工智能	北京-大兴	20-40k	3-5年	硕士	康龙化成	医药外包	沪深A股上
AI人工智能	北京-昌平	20-40k	3-5年	硕士	康龙化成	医药外包	沪深A股上
AI人工智能	北京-通州	20-40k	3-5年	硕士	康龙化成	医药外包	沪深A股上
AI人工智能	北京-海淀	30-60k	经验不限	博士	康迈迪森	专业技术服	A轮
人工智能	北京	25-30k	1-3年	统招本科	中移雄安信通信设备	融资未公开	1000-2000
人工智能	北京	20-21k	5-10年	统招本科	某北京通信通信设备	已上市	10000人以
人工智能	北京	20-40k	3-5年	本科	某北京通信通信设备	已上市	10000人以
AI算法工程	北京-大兴	30-55k	1-3-5年	博士	某北京医疗医疗机构	100-499人	https://www.liepin.com/a/49516913.shtml?pgRef=c_pc
AI算法总监	北京-顺义	150-180k	5-10年	硕士	某知名公司智能硬件	融资未公开	100-499人
AI算法总监	北京	50-70k	5-10年	硕士	某北京智能智能硬件	融资未公开	100-499人
AI算法总监	北京	60-90k	5-10年	本科	某智能硬件智能硬件	融资未公开	100-499人
AI算法总监	北京	60-90k	5-10年	本科	某北京智能智能硬件	融资未公开	100-499人
北京量化私	北京	50-80k	1-3-3年	博士	某北京基金基金/证券/	不需要融资	50-99人
北京量化私	北京	50-80k	1-3-3年	博士	某北京基金基金/证券/	不需要融资	50-99人
北京量化私	北京	70-100k	1-3-3年	博士	某北京基金基金/证券/	不需要融资	50-99人
AI算法软件	北京-海淀	12-24k	1-3年	硕士	中科海拓(中)专业技术服	融资未公开	50-99人
AI算法助理	北京	20-35k	经验不限	博士	清华大学电子/半导体	100-499人	https://www.liepin.com/job/1944509691.shtml?pgRef=
医学人工智	北京-海淀	25-45k	经验不限	博士	神州医疗	IT服务	100-499人
AI算法首席	北京	80-110k	1-3-5年	博士	北京宜人销批发/零售	1-49人	https://www.liepin.com/job/1940777147.shtml?pgRef=

数据爬取过程中有些公司有融资情况列，因此“企业规模”和“融资情况”同时爬取，之后将“融资情况”去除只保留“企业规模”列

```
# 保存合并后的数据框
merged_df = pd.concat([df1, df2], axis=1)

# 添加列名
merged_df.columns = ['岗位名称', '区位', '薪资', '工作年限', '学历', '公司名', '企业类别', '融资情况', '企业规模']

# 将 '融资情况' 列中的 NaN 值替换为 '无'
merged_df['融资情况'] = merged_df.apply(lambda row: '无' if pd.isna(row['融资情况']) else row['融资情况'], axis=1)

# 将 '融资情况' 列中不包含关键字 "人" 的数据项替换为空值
merged_df['融资情况'] = merged_df.apply(lambda row: '' if '人' not in row['融资情况'] else row['融资情况'], axis=1)

# 合并 '融资情况' 和 '企业规模'（第九）列
merged_df['融资情况_企业规模'] = merged_df['融资情况'].fillna('') + ' ' + merged_df['企业规模'].fillna('')

# 删除原始的 '融资情况' 和 '企业规模' 列
merged_df = merged_df.drop(['融资情况', '企业规模'], axis=1)
```

详情页文件示例（2列）

关键词列表	岗位要求
-------	------

	A	B
1	算法研究 算法开发 Python PyTorch	工作职责 1、进行人工智能算法研究和开发，负责AI算法选型、AI训练工具选择使用以及AI训练环境搭建等工作。 2、进行AI训练数据的筛选和处理，对数据进行预处理、清洗和转换，确保数据的质量和适用性。 3、负责AI训练结果的打标和标注，以提高质量。

岗位名称	区位	薪资	工作年限	学历	关键词（列表）	公司名	企业类别
------	----	----	------	----	---------	-----	------

需要将“关键词列表”放到“学历”列之后

```
# 交换 '融资情况_企业规模' 和 '岗位要求' 列
merged_df = merged_df[['岗位名称', '区位', '薪资', '工作年限', '学历', '关键词', '公司名', '企业类别', '企

# 更新列名
merged_df.columns = ['岗位名称', '区位', '薪资', '工作年限', '学历', '关键词', '公司名', '企业类别', '企

# 加上更新时间列，统一填写20231225
merged_df['更新时间'] = '2023-12-25'

# 保存合并后的文件
merged_df.to_csv('猎聘_20231225.csv', index=False, encoding='utf-8')
```

2. z合并详情页

大致页文件示例（7列）

岗位名称	地区/区位	薪资	工作年限	学历	关键词列表	详情页链接
------	-------	----	------	----	-------	-------

	A	B	C	D	E	F	G	H	I	J	K	L
1	ai算法工程北京-朝阳		2万-3.5万 • 14薪	1-3年	硕士	<div class="iteminfo_line3_welfare"><div	http://jobs.zhaopin.com/CC586014530J40543527614.htm?refco					
2	ai算法工程北京-丰台		1.2万-1.5万	1-3年	大专	<div class="iteminfo_line3_welfare"><div	http://jobs.zhaopin.com/CCL1452832710J40502556216.htm?ref					
3	AI算法研究北京-海淀		1.5万-3万	经验不限	本科	<div class="iteminfo_line3_welfare"><div	http://jobs.zhaopin.com/CC649781480J40506823505.htm?refco					
4	AI算法工程北京-海淀		2万-3.5万	经验不限	硕士	<div class="iteminfo_line3_welfare"><div	http://jobs.zhaopin.com/CCL1340884310J40467696404.htm?ref					
5	视频AI算法北京-海淀		1.5万-3万 • 13薪	经验不限	本科	<div class="iteminfo_line3_welfare"><div	http://jobs.zhaopin.com/CC120911470J40591967415.htm?refco					
6	ai算法工程北京-海淀		1.5万-1.6万	1年以下	本科	<div class="iteminfo_line3_welfare"><div	http://jobs.zhaopin.com/CCL1272782040J40560750914.htm?ref					

把"关键词列表"HTML的数据转换成关键词列表

```

from bs4 import BeautifulSoup
import pandas as pd

# 读取第一个CSV文件
df1 = pd.read_csv('大致页数据0102.csv')
# # 加上列名
# df1.columns = ['岗位名称', '区位', '薪资', '工作年限', '学历', '关键词1', '关键词2', '关键词3', '链接']

# 定义一个函数来解析 HTML 数据并提取文本
def parse_html(html):
    if pd.notna(html) and isinstance(html, str):
        soup = BeautifulSoup(html, 'html.parser')
        items = soup.find_all('div', class_='iteminfo_line3_welfare_item')
        return [item.get_text() for item in items]
    else:
        return []

# 将 HTML 数据的第六列应用解析函数并覆盖原始数据
df1.iloc[:, 5] = df1.iloc[:, 5].apply(parse_html)

# 丢弃原始的'链接'列
# 删除最后一列
df1 = df1.drop(df1.columns[-1], axis=1)

```

	A	B	C	D	E	F
1	邦邦汽服	互联网, 互	500-999人	职位描述 图像处理 OCR识别 TensorFlo wPyTorch 视觉图像 算法视频 算法 Python 主要工作 内容 1、负责计 算机视觉 (图像、 OCR) 算法 设计及优 化、模型 训练与测 试、工程 化部署; 2、负责图 像分类、 图像分割 、图像特 征工程、 ocr识别中 的两个或 两个以上 的方式	更新于 今天	

```
# 读取第二个CSV文件
df2 = pd.read_csv('详情页数据0102.csv')
# # 加上列名
# df2.columns = ['公司名', '企业类别', '企业规模', '岗位要求', '更新时间']

# 保存合并后的数据框
merged_df = pd.concat([df1, df2], axis=1)

# 保存合并后的文件
merged_df.to_csv('智联0102.csv', index=False)
```

第二步：xyz数据格式统一

1. 地区格式：地区-区位（如：北京-海淀）

```
import pandas as pd

# # 读取Excel文件，替换 'your_excel_file.xlsx' 为实际文件路径
# df = pd.read_excel('your_excel_file.xlsx')

def clean_location(location):
    if len(location.split('.'))>=2:
```

```

        location = location.split('.')[0] + '-' + location.split('.')[1]
        location = location.replace('区', '')
    elif len(location.split('-'))>=2:
        location = location.replace('区', '')
    return location

# # 应用处理函数到"区位"列
# df['区位'] = df['区位'].apply(clean_location)

# # 保存修改后的文件
# df.to_excel('cleaned_data.xlsx', index=False)
ex1 = "北京"
ex2 = "北京-石景山区"
ex3 = "北京·昌平区·回龙观"
ex4 = "北京·昌平区·CBD"
ex5 = "北京·大兴区"

print(f"原始地区：{ex1}, 转换后地区：{clean_location(ex1)}")
print(f"原始地区：{ex2}, 转换后地区：{clean_location(ex2)}")
print(f"原始地区：{ex3}, 转换后地区：{clean_location(ex3)}")
print(f"原始地区：{ex4}, 转换后地区：{clean_location(ex4)}")
print(f"原始地区：{ex5}, 转换后地区：{clean_location(ex5)}")

```

2. 学历格式：把"统招本科"统一成"本科"

```

import pandas as pd

# 读取Excel文件，替换 'your_excel_file.xlsx' 为实际文件路径
df = pd.read_excel('your_excel_file.xlsx')

# 处理"学历"列
def clean_education(education):
    if '统招本科' in education:
        return '本科'
    else:
        return education

# 应用处理函数到"学历"列
df['学历'] = df['学历'].apply(clean_education)

# 保存修改后的文件
df.to_excel('cleaned_data.xlsx', index=False)

```

3. 工作年限格式

```

# 1年以内、一年以下-->1年以下
# 无经验 --> 经验不限
# 应届生保留

import pandas as pd

# 读取Excel文件，替换 'your_excel_file.xlsx' 为实际文件路径

```

```

df = pd.read_excel('your_excel_file.xlsx')

# 处理"工作年限"列
def clean_experience(experience):
    if '以内' in experience or '以下' in experience:
        return '1年以下'
    elif '无经验' in experience:
        return '经验不限'
    elif '应届生' in experience:
        return '应届生'
    else:
        return experience

# 应用处理函数到"工作年限"列
df['工作年限'] = df['工作年限'].apply(clean_experience)

# 保存修改后的文件
df.to_excel('cleaned_data.xlsx', index=False)

```

4. 更新时间：年-月-日 (xxxx-xx-xx)

```

import pandas as pd
import re

# # 读取修改后的文件
# merged_df = pd.read_csv('modified_merged_file.csv')
# 处理更新时间列
def convert_update_date(date_str):
    # Check if the date string contains "今天"
    if '今天' in date_str:
        # Replace "今天" with the current date
        date_str = date_str.replace('今天', pd.to_datetime('today').strftime('%m月%d日'))

    # Extract the month and day from the string using regular expression
    match = re.search(r'(\d+)月(\d+)日', date_str)

    if match:
        month, day = map(int, match.groups())

        # Assume the year is the current year (you can replace this with a specific year if n
        current_year = pd.to_datetime('today').year

        # Format the date as "更新于：年-月-日"
        formatted_date = f"{current_year}-{month:02d}-{day:02d}"
        return formatted_date
    else:
        updated_date_str = date_str.replace("更新于:", "")
        return updated_date_str

# merged_df['更新时间'] = merged_df['更新时间'].apply(convert_update_date)
# # 保存修改后的文件
# merged_df.to_csv('final_merged_file.csv', index=False)
ex1 = "更新于 12月4日"

```



```

ex2 = "更新于:2023-12-10"
ex3 = "更新于12月8日"
ex4 = "更新于 今天"

print(f"原始:{ex1}, 转换后:{convert_update_date(ex1)}")
print(f"原始:{ex2}, 转换后:{convert_update_date(ex2)}")
print(f"原始:{ex3}, 转换后:{convert_update_date(ex3)}")
print(f"原始:{ex4}, 转换后:{convert_update_date(ex4)}")

```

5. 薪资格式：

10k-14k-14薪

10k-14k

保留

"xxxx-xxxx元/周"
 "xxx-xxx元/时"
 "xxx-xxx/天"

```

def convert_salary(salary):
    salary = salary.strip()
    if '面议' in salary:
        return '薪资面议'
    elif '急聘' in salary:
        return '急聘'

    # 保留实习岗位描述, '/'天'映射为'元/天'
    special_cases = {
        '元/周': '元/周',
        '元/时': '元/时',
        '/天': '元/天',
        '元/次': '元/次'
    }

    for key, value in special_cases.items():
        if key in salary:
            salary = salary.replace(key, value)

    # 如果有多少薪, 按照·拆分
    parts = salary.split('.')
    base_salary = parts[0].strip().replace('K', '')
    multiplier = ''

    if len(parts) > 1:
        multiplier = '.' + parts[1].strip()

    # 对于“元/月”描述的数据的处理
    if '元/月' in salary:
        try:
            # “1万”转换为“10k”
            if '万' in base_salary:
                base_salary = base_salary.replace('万', '')
                min_salary, max_salary = map(lambda x: int(float(x.replace('元/月', ''))*10),
                                              base_salary.split('-'))
                return f"{min_salary}k-{max_salary}k"

```

```

        else:
            min_salary, max_salary = map(lambda x: int(int(x.replace('元/月', ''))/1000), salary.split('-'))
            return f"{min_salary}k-{max_salary}k{multiplier}"
    except ValueError:
        return salary
else:
    try:
        min_salary, max_salary = map(lambda x: int(float(x.replace('万', ''))* 10), salary.split('-'))
        return f"{min_salary}k-{max_salary}k{multiplier}"
    except ValueError:
        # "1万"转换为"10k", "1千"转换为"1k"
        if '万' in base_salary:
            if '千' in base_salary:
                min_salary = base_salary.split('-')[0].replace('千', '')
                max_salary = int(float(base_salary.split('-')[1].replace('万', ''))*10)
                return f"{min_salary}k-{max_salary}k"
            else:
                min_salary, max_salary = map(lambda x: int(float(x.strip()).replace('万', '')), salary.split('-'))
                return f"{min_salary}k-{max_salary}k"
        elif '千' in base_salary:
            min_salary, max_salary = map(lambda x: int(float(x.strip()).replace('千', '')), salary.split('-'))
            return f"{min_salary}k-{max_salary}k"
        else:
            return salary

# 示例调用 convert_salary 函数
example_salary1 = "70-90K·14薪"
example_salary2 = "2万-4万·15薪"
example_salary3 = "70-90K"
example_salary4 = "xxxx-xxxx元/周"
example_salary5 = "xxx-xxx元/时"
example_salary6 = "xxx-xxx/天"
example_salary7 = "面议"
example_salary8 = "急聘"
example_salary9 = "14000-15000元/月"
example_salary10 = "8000-12000元/月"
example_salary11 = "2万-4万"
example_salary12 = "\r\n 1.3万-1.8万"
example_salary13 = "\r\n1.2万-1.3万          · 13薪"
example_salary14 = "1.5万-2.5万元/月"
example_salary15 = "5千-8千"
example_salary16 = "5千-1.3万"
example_salary17 = "30-60k·14薪"

```

6. 企业规模格式

```

import pandas as pd

def clean_company_size(size):
    # 只有在 size 是字符串时才执行包含 '20人以下' 的检查。这样可以避免对浮点数执行不可迭代的 in 操作。
    size = '无' if pd.isna(size) else size
    size = size.strip()

```

```

if '20人以下' in size:
    # 因为企业规模中有空值nan，nan属于浮点数，所以会出现这样的情况（对浮点数执行不可迭代的 in 操作）
    # 建议把nan直接换成字段“无”，方便后续处理
    return '0-20人'
else:
    return size

```

boss	猎聘	智联	统一格式
0-20人	1-49人	20人以下	0-20人
20-99人	50-99人	20-99人	保留
100-499人	100-499人	100-299人	保留
500-999人	500-999人	300-499人	保留
1000-9999人	1000-2000人	500-999人	保留
10000人以上	2000-5000人	1000-9999人	保留
(空白)	5000-10000人	(空白)	保留
	10000人以上		保留
	(空白)		保留

7. z关键词进一步转换

```
"['CNN', 'TensorFlow', 'PyTorch', '数据开发', '人工智能、视觉识别', '深度学习算法', 'Python']"
```

'CNN'

'TensorFlow'

'PyTorch'

'数据开发'

'人工智能、视觉识别'

'深度学习算法'

'Python'

```

def format_string_to_newline(data):
    # 去除字符串中的引号并替换逗号为换行符
    formatted_data = data.replace("[", "").replace("]", "").replace("'", "").replace(", ", "\n")

    return formatted_data

# 示例用法
data_str = "['CNN', 'TensorFlow', 'PyTorch', '数据开发', '人工智能、视觉识别', '深度学习算法', 'Python']"
result_str_to_newline = format_string_to_newline(data_str)

print(result_str_to_newline)

```

▼ 数据格式统一完整代码

```

import pandas as pd
import re
import chardet

# 地区格式：北京·昌平区·回龙观-->北京-昌平
def clean_location(location):

```

```

    if len(location.split('.'))>=2:
        location = location.split('.')[0] + '-' + location.split('.')[1]
        location = location.replace('区', '')
    elif len(location.split('-'))>=2:
        location = location.replace('区', '')
    return location

# 薪资格式：50-80K·13薪（“薪资面议”，“面议”-->“薪资面议”）（保留“急聘”）
def convert_salary(salary):
    if '面议' in salary:
        return '薪资面议'
    elif '急聘' in salary:
        return '急聘'

    # Mapping for special cases
    special_cases = {
        '元/周': '元/周',
        '元/时': '元/时',
        '/天': '元/天',
        '元/次': '元/次'
    }

    for key, value in special_cases.items():
        if key in salary:
            salary = salary.replace(key, value)

    parts = salary.split('.')
    base_salary = parts[0].strip().replace('K', '')
    multiplier = ''

    if len(parts) > 1:
        multiplier = '.' + parts[1].strip()

    if '元/月' in salary:
        try:
            if '万' in base_salary:
                base_salary = base_salary.replace('万', '')
                min_salary, max_salary = map(lambda x: int(float(x.replace('元/月', ''))*10),
                                              base_salary.split('-'))
                return f"{min_salary}k-{max_salary}k"
            else:
                min_salary, max_salary = map(lambda x: int(int(x.replace('元/月', ''))/1000),
                                              base_salary.split('-'))
                return f"{min_salary}k-{max_salary}k{multiplier}"
        except ValueError:
            return salary
    else:
        try:
            min_salary, max_salary = map(lambda x: int(float(x.replace('万', ''))*10),
                                          base_salary.split('-'))
            return f"{min_salary}k-{max_salary}k{multiplier}"
        except ValueError:
            if '万' in base_salary:
                if '千' in base_salary:
                    min_salary = base_salary.split('-')[0].replace('千', '')
                    max_salary = int(float(base_salary.split('-')[1].replace('万', ''))*10)
                    return f"{min_salary}k-{max_salary}k"

```

```

        else:
            min_salary, max_salary = map(lambda x: int(float(x.strip()).replace('万',
            return f"{min_salary}k-{max_salary}k"
        elif '千' in base_salary:
            min_salary, max_salary = map(lambda x: int(float(x.strip()).replace('千', ''))
            return f"{min_salary}k-{max_salary}k"
        else:
            return salary

# 更新时间格式: year-month-day(yyyy-mm-dd)
def convert_update_date(date_str):
    # Check if the date string contains "今天"
    if '今天' in date_str:
        # Replace "今天" with the current date
        date_str = date_str.replace('今天', pd.to_datetime('today').strftime('%m月%d日'))

    # Extract the month and day from the string using regular expression
    match = re.search(r'(\d+)月(\d+)日', date_str)

    if match:
        month, day = map(int, match.groups())

        # Assume the year is the current year (you can replace this with a specific year if n
        current_year = pd.to_datetime('today').year

        # Format the date as "更新于:年-月-日"
        formatted_date = f"{current_year}-{month:02d}-{day:02d}"
        return formatted_date
    else:
        updated_date_str = date_str.replace("更新于:", "")
        return updated_date_str

# 工作年限统一: 1年以内、一年以下-->1年以下
def clean_experience(experience):
    if '以内' in experience or '以下' in experience:
        return '1年以下'
    elif '无经验' in experience:
        return '经验不限'
    elif '应届生' in experience:
        return '应届生'
    else:
        return experience

# 学历统一: 统招本科 --> 本科
def clean_education(education):
    if '统招本科' in education:
        return '本科'
    else:
        return education

# 文件编码格式统一: utf-8
def detect_encoding(file_path):
    with open(file_path, 'rb') as f:

```

```

        result = chardet.detect(f.read())
        encoding = result['encoding'].lower()
        if encoding == 'gb2312':
            encoding = 'gb18030' # Use 'gb18030' instead of 'gb2312'
        return encoding

# 企业规模统一：20人以下 --> 0-20人
def clean_company_size(size):
    # 把nan直接换成字段“无”，方便后续处理
    size = '无' if pd.isna(size) else size
    size = size.strip()
    if '20人以下' in size:
        return '0-20人'
    else:
        return size

csv_file_path = '智联0102.csv'
# 检测文件编码
file_encoding = detect_encoding(csv_file_path)
# 输出检测到的编码
print(f"Detected encoding: {file_encoding}")
# 读取CSV文件
df = pd.read_csv(csv_file_path, encoding=file_encoding)

# 检查数据有几列
print(f"Number of columns in the DataFrame: {df.shape[1]}")

# 添加列名
df.columns = ['岗位名称', '区位', '薪资', '工作年限', '学历', '关键词', '公司名', '企业类别', '企业规模']

# 统一处理列格式# 删除岗位要求列中的空白项
df = df.dropna(subset=['岗位要求'])
df['区位'] = df['区位'].apply(clean_location)
df['薪资'] = df['薪资'].apply(convert_salary)
df['更新时间'] = df['更新时间'].apply(convert_update_date)
df['工作年限'] = df['工作年限'].apply(clean_experience)
df['学历'] = df['学历'].apply(clean_education)
df['企业规模'] = df['企业规模'].apply(clean_company_size)
# 针对z
# def format_string_to_newline(data):
#     # 去除字符串中的引号并替换逗号为换行符
#     formatted_data = data.replace("[", "").replace("]", "").replace("'", "").replace(", ",
#
#     return formatted_data
# df['关键词'] = df['关键词'].apply(format_string_to_newline)

# 新增“来源”列并填充数据
df['来源'] = '智联'

# 保存修改后的文件
df.to_csv('unified_智联_20240102.csv', index=False, encoding='utf-8')

```

第三步：xyz数据合并

1. 每日数据合并、删除重复项（所有行删除重复项） merged_data_0104以日期结尾

```
import glob
import pandas as pd

def merge_csv(filenamees):
    """Merge CSV files."""
    frames = [pd.read_csv(filename, encoding='utf-8') for filename in filenamees]
    merged_data = pd.concat(frames, ignore_index=True)
    # 删除 '关键词1'、'关键词2'、'关键词3' 列
    # merged_data = merged_data.drop(['关键词1', '关键词2', '关键词3'], axis=1)
    return merged_data

if __name__ == '__main__':
    # Example filenames, replace with your actual file paths
    filenames = ['data_merge\\merged_data_1226.csv', 'data_merge\\merged_data_1227.csv', 'data_me

    # Merge data
    merged_data = merge_csv(filenamees)

    # 总共合并的数据项
    print(f'Total merged data: {len(merged_data)} rows')

    # Assuming you want to drop duplicates based on all columns
    merged_data_no_duplicates = merged_data.drop_duplicates()

    # 删除数据后的数据总和
    print(f'Total data after removing duplicates: {len(merged_data_no_duplicates)} rows')

    # Save merged and deduplicated data to a new CSV file
    merged_data_no_duplicates.to_csv('merged_data_0104.csv', index=False, encoding='utf-8')
```

2. 所有数据合并

- a. 合并并删除所有行重复项（代码同上） merged_data_all 以all结尾
- b. 合并并删除（不含“更新时间”、“来源”）剩余所有行重复项 merged_data_final 以final结尾

```
import pandas as pd

# 读取包含12列数据的 CSV 文件
df = pd.read_csv('merged_data_all.csv')

# 打印删除前的数据项总数
print(f'删除前的数据项总数: {len(df)}')

# 删除前10列重复的数据项
df_no_duplicates = df.drop_duplicates(subset=df.columns[:10])

# 打印删除后的数据项总数
print(f'删除后的数据项总数: {len(df_no_duplicates)}')
```

```
# 保存删除重复项后的数据到新的 CSV 文件  
df_no_duplicates.to_csv('merged_data_final.csv', index=False)
```