内容简介

数据检视主要是通过对数据质量以及字段情况进行了解，由于本案例数据集已经经过处理，不存在缺失异常重复等情况，因此不需要进行数据清理，只需要根据建模的需要进行必要的数据转换即可，同时也省去了数据改造方案，具体内容如下：

1. 主要内容

- 数据质量和字段情况；
- 个人特征与通过岗位数的相关性探索；
- 数据转换。

2. 学习目标

学完本节，能解决以下问题：

- 如何通过 statsmodels 库中的方法 anova_lm 实现单因素方差分析？

方差分析解释

在本案例中，通过方差分析可以检验学历和通过岗位数是否具有显著相关性。通过类 statsmodels.stats.anova 中的方法 anova_lm 可以实现单因素方差分析，结果中 F 表示 F 统计量（或 F 值），PR(>F)表示 P 值，一般来说 P 值小于 0.05 表明具有显著相关性。

案例中单因素方法分析结果：

	df	sum_sq	mean_sq	F	PR(>F)
edu	3.0	427.66581	142.555270	47.041005	1.960523e-28
Residual	996.0	3018.32519	3.030447	NaN	NaN

P 值远远小于 0.05，表明学历和通过岗位数存在显著相关性。

为了说明表中每一数值的具体计算公式，定义现有数据的结构如下：

观测值（j）	因素（A_i）			
	水平 A_1	水平 A_2	…	水平 A_k
1	x_{11}	x_{12}	…	x_{1k}
2	x_{21}	x_{22}	…	x_{2k}
…	…	…	…	…
n	x_{n1}	x_{n2}	…	x_{nk}

k 表示因素水平的个数，n 表示全部观察值的个数。

通过方法 anova_lm 可以得到单因素方法分析的结果如下表所示：

方差来源	df（自由度）	sum_sq（平方和）	mean_sq（均方）	F（F 统计量）	P 值
组间	k-1	SSA	MSA	MSA/MSE	
组内	n-k	SSE	MSE		

表中的第一、二行分别描述的是与组间方差、组内方差相关的数据量。df 表示自由度，指的是在计算对应方差时，取值不受限制的变量个数。

SSA（Sum of Squares for Factor A）是每个水平的均值与总平均值的离差平方和，

又称组间平方和，计算公式如下：

$$SSA = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(\bar{x}_i - \bar{x})^2 = \sum_{i=1}^{k} n_i(\bar{x}_i - \bar{x})^2$$

SSE（Sum of Squares for Error）是每个水平的各样本数据与该组均值的离差平方和，又称组内离差平方和，计算公式如下：

$$SSE = \sum_{i=1}^{k}\sum_{j=1}^{n_i}(x_{ij} - \bar{x}_i)^2$$

MSA（Mean of Squares for Factor A）与 MSE（Mean of Squares for Error）分别表示组间方差与组内方差，计算公式为：

$$MSA = \frac{SSA}{k-1}$$
$$MSE = \frac{SSE}{n-k}$$

而 F 统计量即为 MSA/MSE，一般来说，若 P 值小于 0.05 表明各水平中的观察值均值具有显著相关性。因此，在单因素方差分析中，主要关注最后的 P 值即可。

关于类 statsmodels.stats.anova 中的方法 anova_lm 的单因素方差分析，其具体使用方法与含义解释可以参考官方文档：

http://www.statsmodels.org/devel/generated/statsmodels.stats.anova.anova_lm.html

其中方法 ols 是通过公式的形式定义单因素方差分析中的因素水平与观察值，以及具体的数据来源，具体公式的形式与含义可以参考链接。

http://www.statsmodels.org/devel/example_formulas.html#categorical-variables

你需要在作业中继续完成英语水平、工作经验与岗位通过数之间的相关性探索。

数据检视与转换总结

1、如何通过 statsmodels 库中的方法 anova_lm 实现单因素方差分析？

单因素方差分析是从结果中的 P 值判断某因素与观察值的相关性，一般来说 P 值小于 0.05 表明该因素与观察值具有显著相关性。这里以个人特征学历为例，判断其与通过岗位数之间是否具有显著相关性：

看过官方文档，我们知道 anova_lm()括号中接受一个或多个线性模型，我们通过类 statsmodels.formula.api 中方法 ols 构建一个关于<学历-通过岗位数>的线性模型，即可得到两者之间的相关性 P 值，例如案例中的代码：

```
from statsmodels.stats.anova import anova_lm
from statsmodels.formula.api import ols

res = ols(formula='passed_num~edu', data=candidate).fit()
table = anova_lm(res)
table
```

	df	sum_sq	mean_sq	F	PR(>F)
edu	3.0	427.66581	142.555270	47.041005	1.960523e-28
Residual	996.0	3018.32519	3.030447	NaN	NaN

其中 ols 模型中的参数 formula 取值为观察值和因素，之间用“~”分开，参数 data 取值为观测值和因素来源的 DataFrame 数据集。得到的单因素方差分析结果如图中所示，表中每一个取值代表的意义如下表所示：

方差来源	df（自由度）	sum_sq（平方和）	mean_sq（均方）	F（F 统计量）	P 值
组间	k-1	SSA	MSA	MSA/MSE	
组内	n-k	SSE	MSE		

其中 P 值小于 0.05，因此原假设被证实，即学历和岗位通过数之间具有显著相关性。

内容简介

本节将使用转换后的数据，分别构建基于用户的协同过滤模型和基于物品的协同过滤模型，为目标岗位推荐合适的应聘者。

1. 主要内容

- 协同过滤算法介绍；
- 基于用户的协同过滤算法；
- 基于物品的协同过滤算法。

2. 学习目标

学完本节，能解决以下问题：

- 为了回答主业务问题，如何分别基于用户和物品构建协同过滤模型？

协同过滤算法

当下主流的推荐算法是协同过滤方法，其中协同过滤最典型的两个分支是基于用户的协同过滤和基于物品的协同过滤，两种方法都需要计算相似度，最常使用的一种相似度计算方法是余弦相似度。

1. 余弦相似度

余弦相似度计算的是两个向量夹角的余弦函数，即对于向量 **A** = (A_1, A_2, ..., A_n)和向量 **B** = (B_1, B_2, ..., B_n)，两个向量的余弦相似度。

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

cos(θ)的值越大，说明两个用户/物品越相似。

2. 基于用户的协同过滤算法

基于用户的协同过滤算法是通过用户的历史行为数据(如商品购买，收藏，内容评论或分享)发现用户对物品的喜好，并根据这些喜好得到用户对物品的评分矩阵。再根据用户的评分向量计算用户之间的相似度，并在相似度高的用户间进行物品推荐。

在给定目标用户的条件下，具体步骤包括：

1）计算其他用户和目标用户的相似度，本次案例中相似度使用余弦相似度。

2）找出目标用户 k 个最相似的邻居。

3）针对这些邻居喜欢的物品，根据邻居与目标用户的相似度计算出每个物品的推荐度。

4）根据每一件物品的推荐度高低给用户推荐物品。

在本次案例中，算法中的用户对应岗位，物品对应应聘者。岗位之间的相似度主要是

通过岗位对应聘者的操作记录得到，即应聘者通过岗位的信息。根据应聘者岗位通过矩阵计算岗位相似度，然后给目标岗位推荐相似岗位通过的应聘者。

3. 基于物品的协同过滤算法

基于物品的协同过滤算法和基于用户的系统过滤相似，也是在用户对物品的评分矩阵的基础上，计算物品之间的相似度，然后给用户推荐与他/她购买过的物品相似的物品。

在给定目标用户的条件下，具体步骤包括：

1）计算物品之间的相似度。

2）根据物品之间的相似度以及用户购买历史行为给用户生成推荐列表。

本次案例中，用户对应岗位，物品对应应聘者。应聘者之间的相似度可以有两种计算方法：基于应聘者个人信息，或者基于应聘者通过岗位信息。在实际的推荐系统中，通常都是采用综合不同推荐方法结果来实现的，在本次案例中我们分别基于以上两种方案实现基于物品的协同过滤算法。

推荐算法的评价方式很多，从业务的角度来看包括：用户满意度、预测准确度、商品覆盖率、推荐多样性、推荐新颖性、用户信任度等等。

协同过滤算法总结

1、为了回答主业务问题，如何分别基于用户和物品构建协同过滤模型？

（1）协同过滤算法

当下主流的推荐算法是协同过滤方法，其中协同过滤最典型的两个分支是基于用户的协同过滤和基于物品的协同过滤，两种方法都需要计算相似度，最常使用的一种相似度计算方法是余弦相似度。

注：余弦值越大，说明两个用户/物品越相似。

（2）基于用户的协同过滤算法

通过用户的历史行为数据(如商品购买，收藏，内容评论或分享)发现用户对物品的喜好，并根据这些喜好得到用户对物品的评分矩阵。再根据用户的评分向量计算用户之间的相似度，并在相似度高的用户间进行物品推荐。

在给定目标用户的条件下，具体步骤包括：

1）计算其他用户和目标用户的相似度。

2）找出目标用户 k 个最相似的邻居。

3）针对这些邻居喜欢的物品，根据邻居与目标用户的相似度计算出每个物品的推荐度。

4）根据每一件物品的推荐度高低给用户推荐物品。

基于上面的步骤，在本次案例中，用户对应岗位，物品对应应聘者。首先计算用户之间的余弦相似度，

余弦相似度可以通过 sklearn.metrics.pairwise 中的 cosine_similarity 方法计算得到，代码及结果如下：

```
# 岗位之间的余弦相似度
from sklearn.metrics.pairwise import cosine_similarity
jobsim = pd.DataFrame(cosine_similarity(job_cdt_passed),
                      index=np.arange(1,51),columns=np.arange(1,51))
jobsim
```

	1	2	3	4	5	6	7	8	9	10	...	41	42	43	44	45
1	1.000000	0.173687	0.146845	0.139904	0.220510	0.183136	0.202147	0.222361	0.136483	0.204942	...	0.178478	0.182768	0.142107	0.171632	0.178478
2	0.173687	1.000000	0.158555	0.156269	0.151387	0.206417	0.218454	0.132089	0.184697	0.162650	...	0.200528	0.152236	0.201058	0.194105	0.205805
3	0.146845	0.158555	1.000000	0.183010	0.189956	0.132592	0.230988	0.188990	0.179949	0.162471	...	0.207215	0.189859	0.191360	0.161576	0.136325
4	0.139904	0.156269	0.183010	1.000000	0.170331	0.164771	0.190059	0.175055	0.155857	0.176141	...	0.166248	0.222229	0.213568	0.196403	0.223395
5	0.220510	0.151387	0.189956	0.170331	1.000000	0.142520	0.150550	0.181699	0.194127	0.177286	...	0.140203	0.198478	0.221673	0.159780	0.226482
6	0.183136	0.206417	0.132592	0.164771	0.142520	1.000000	0.166058	0.192842	0.127975	0.137199	...	0.144668	0.193728	0.156208	0.170554	0.150232
7	0.202147	0.218454	0.230988	0.190059	0.150550	0.166058	1.000000	0.204878	0.141874	0.187409	...	0.187477	0.216741	0.177812	0.181199	0.177343
8	0.222361	0.132089	0.188990	0.175055	0.181699	0.192842	0.204878	1.000000	0.228013	0.161380	...	0.202678	0.166336	0.177812	0.227793	0.167209
9	0.136483	0.184697	0.179949	0.155857	0.194127	0.127975	0.141874	0.228013	1.000000	0.183851	...	0.168421	0.219898	0.189974	0.166705	0.163158
10	0.204942	0.162650	0.162471	0.176141	0.177286	0.137199	0.187409	0.161380	0.183851	1.000000	...	0.173036	0.182891	0.135542	0.149174	0.162221
11	0.188482	0.200003	0.152284	0.202083	0.155970	0.172036	0.176878	0.202147	0.194226	0.172583	...	0.125985	0.172324	0.173687	0.171632	0.162730

这里的 job_cdt_passed 是岗位-应聘者通过矩阵，它反映了岗位对应聘者的操作记录，以此计算岗位之间的相似度，而该矩阵可以通过如下代码实现：

```
# 根据passed表生成岗位-应聘者通过矩阵
job_cdt_passed = passed.groupby(['job'])['user'].value_counts().unstack().fillna(0)
job_cdt_passed
```

user	1	2	3	4	5	6	7	8	9	10	...	991	992	993	994	995	996	997	998	999	1000
job																					
1	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	0.0	1.0	0.0	0.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	1.0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	0.0
4	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	0.0	0.0
5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	...	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

每一行为一个岗位，每一列为一个用户；然后通过 query()方法筛选出目标岗位的 k 个邻居（也可以通过其他方法进行筛选），最后基于筛选的出来的邻居岗位 ID，再筛选出通过这些邻居岗位的应聘者，最后从这些应聘者集合中剔除已通过目标岗位的应聘者即可。整体的代码部分可以通过一个函数进行打包，这样就可以根据业务需要针对不同的目标岗位和邻居个数，得到应聘者推荐列表，具体函数定义代码如下：

```
# 基于岗位相似度，给目标岗位推荐应聘者
def recommendByJob(jobId, k):

    # 确定jobId的k个相似岗位
    knb = jobsim[jobId].sort_values(ascending=False)[1:k+1].index
    # 构建应聘者推荐列表
    knb_passed_cdt = passed.query('job in @knb')['user'].unique()

    # 获取jobId已通过应聘者
    passed_cdt = passed.query('job == @jobId')['user'].unique()
    # 从推荐列表中排除已通过应聘者
    recommendList = set(knb_passed_cdt)-set(passed_cdt)

    return recommendList
```

可以演示一下对于 ID 为 5 的岗位，邻居为 3 的情况下推荐的应聘者的代码：

```
# 算法演示
recommendByJob(jobId=5, k=3)
```

```
{1,
 3,
 4,
 5,
 6,
 7,
 11,
 18,
 20,
 25,
 34,
 39,
 41,
 42,
 43,
 46,
 51,
 56,
 57,
 63.
```

（3）基于物品的协同过滤算法

与上一个算法相似，也是在用户对物品的评分矩阵的基础上，计算物品之间的相似度，然后给用户推荐与他/她购买过的物品相似的物品。

在给定目标用户的条件下，具体步骤包括：

1）计算物品之间的相似度。

2）根据物品之间的相似度以及用户购买历史行为给用户生成推荐列表。

本次案例中，用户对应岗位，物品对应应聘者。应聘者之间的相似度可以有两种计算方法：基于应聘者个人信息，或者基于应聘者通过岗位信息。在实际的推荐系统中，通常都

是采用综合不同推荐方法结果来实现的，在本次案例中我们分别基于以上两种方案实现基于物品的协同过滤算法。

```
# 基于应聘者相似度，给目标岗位推荐应聘者
def recommendByCdt(jobId, k, simi='pass'):

    # 获取jobId已通过应聘者
    passed_cdt = passed.query('job == @jobId')['user'].unique()

    # 选择应聘者相似度计算方式
    if simi=='pass':
        cdtsim = cdtsim_pass
    elif simi=='info':
        cdtsim = cdtsim_info
    else:
        print('Please define valid similarity method:"pass" or "info"')

    # 获取通过应聘者的k个相似应聘者，组成推荐列表
    cdt_knb = set()
    for cdt in passed_cdt:
        knb = cdtsim[cdt].sort_values(ascending=False)[1:k+1].index
        cdt_knb = cdt_knb | set(knb)

    # 从推荐列表中排除已通过应聘者
    recommendList = cdt_knb-set(passed_cdt)

    return recommendList
```

基于应聘者个人信息（cdtsim_info）计算应聘者之间相似度的代码如下：

```
# 应聘者之间的余弦相似度-基于应聘者个人信息
cdtsim_info = pd.DataFrame(cosine_similarity(features),
                           index=np.arange(1,1001),columns=np.arange(1,1001))
cdtsim_info
```

	1	2	3	4	5	6	7	8	9	10	...	991	992	993	994	9
1	1.000000	0.597466	0.820050	0.551815	0.582324	0.810770	0.571450	0.515161	0.772124	0.523485	...	0.722204	0.807522	0.384367	0.621028	0.5422
2	0.597466	1.000000	0.540865	0.794429	0.630428	0.520864	0.813748	0.772690	0.582121	0.334845	...	0.409095	0.582270	0.649528	0.669440	0.7810
3	0.820050	0.540865	1.000000	0.529529	0.575912	0.822172	0.541085	0.466238	0.724914	0.547574	...	0.717712	0.805408	0.368996	0.618475	0.5264
4	0.551815	0.794429	0.529529	1.000000	0.615586	0.506853	0.636319	0.774139	0.532548	0.358394	...	0.395319	0.617213	0.684613	0.655521	0.7698
5	0.582324	0.630428	0.575912	0.615586	1.000000	0.569609	0.768014	0.637393	0.523510	0.338258	...	0.563031	0.569922	0.617861	0.783268	0.6336

基于应聘者通过岗位信息计算相似度，首先需要得到应聘者-岗位矩阵:

```
# 根据passed表生成应聘者-岗位通过矩阵
cdt_job_passed = passed.groupby(['user'])['job'].value_counts().unstack().fillna(0)
cdt_job_passed
```

job	1	2	3	4	5	6	7	8	9	10	...	41	42	43	44	45	46	47	48	49	50
user																					
1	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	1.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
3	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	1.0
4	1.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0
5	0.0	1.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	...	0.0	0.0	1.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0

再得到应聘者相似度（cdtsim_pass）：

```
# 应聘者之间的余弦相似度-基于岗位通过数据
cdtsim_pass = pd.DataFrame(cosine_similarity(cdt_job_passed),
                           index=np.arange(1,1001),columns=np.arange(1,1001))
cdtsim_pass
```

	1	2	3	4	5	6	7	8	9	10	...	991	992	993	994	9
1	1.000000	0.385758	0.154303	0.356348	0.241747	0.222375	0.084515	0.202031	0.169031	0.101015	...	0.267261	0.241747	0.119523	0.080582	0.169(
2	0.385758	1.000000	0.250000	0.192450	0.435194	0.240192	0.273861	0.327327	0.091287	0.109109	...	0.096225	0.174078	0.258199	0.174078	0.182
3	0.154303	0.250000	1.000000	0.096225	0.087039	0.160128	0.273861	0.327327	0.091287	0.218218	...	0.000000	0.435194	0.000000	0.261116	0.182
4	0.356348	0.192450	0.096225	1.000000	0.301511	0.092450	0.105409	0.125988	0.105409	0.251976	...	0.222222	0.201008	0.149071	0.100504	0.000(
5	0.241747	0.435194	0.087039	0.301511	1.000000	0.334497	0.286039	0.113961	0.000000	0.227921	...	0.201008	0.272727	0.134840	0.090909	0.286(

注：

推荐算法的评价方式很多，从业务的角度来看包括：用户满意度、预测准确度、商品覆盖率、推荐多样性、推荐新颖性、用户信任度等等。

本案例主要是对推荐算法的介绍和实现，暂时没有过多的针对结果的解释。