评估结果描述

根据前期的结果看，需要对三个数据集先进性分别处理相应的数据问题，之后在进行数据融合。

数据清洗流程

1. `image_prediction_data` 数据集

该数据集的问题，都是集中在数据质量方面，表现在数据类型和数据格式的问题。但是在整洁度来看，暂时没有发现问题。

- `tweet_id` 是 `int64` 的整数类型，实际应当保存为 `object` 类型

  > 该问题的解决直接通过 `astype` 的方法对数据类型进行了转换

- `p1` 、 `p2` 以及 `p3` 数据中首字母存在大写和小写混用的情况

  > 这三个 `field` 的数据直接通过 `applymap` 方法，以 `lambda` 函数穿件了一个对数据的字符串进行转换的方式来完成的转换:

```
image_data_copy[["p1", "p2", "p3"]] = \
image_data_copy[["p1", "p2", "p3"]].applymap(lambda x: x.title())
```

- `p1_conf` 、 `p2_conf` 以及 `p3_conf` 是比例数据，可以确认需要保留的小数点位数

  > 这三个 `field` 的数据同样是通过 `applymap` 方法，以 `lambda` 函数穿件了一个对数据的字符串进行转换的方式来完成的转换:

```
image_data_copy[["p1_conf", "p2_conf", "p3_conf"]] = \
    image_data_copy[["p1_conf", "p2_conf", \
        "p3_conf"]].applymap(lambda x: round(x, 4))
```

2. `twitter_archive_data` 数据集

该数据的同样存在数据类型不合适、数据缺失的数据质量问题，同时在数据整洁度上也同样存在问题：

- `tweet_id` 是 `int64` 的整数类型，实际应当保存为 `object` 类型

  > 该问题的解决直接通过 `astype` 的方法对数据类型进行了转换

- `in_reply_to_status_id`、`in_reply_to_user_id`、`retweeted_status_id` 以及 `retweeted_status_user_id` 是 `float64` 的整数类型，实际应当保存为 `object` 类型

  > 通过 `applymap` 的方法进行数据类型转换：

```
twitter_enhanced_data_copy[["in_reply_to_status_id", "in_reply_to_user_id",
twitter_enhanced_data_copy[["in_reply_to_status_id", "in_reply_to_user_id",
lambda x: str(x))
```

- `timestamp` 以及 `retweeted_status_timestamp` 是 `object` 的对象数据(字符串模式)，实际应当保存为 `datetime` 类型

  > 这两个 `field` 的转换也是通过 `apply` 的方法分别对各个 `field` 进行了转换。另外需要注意该方法中传入的方法是使用的 `pd.Timestamp` 方法

```
twitter_archive_data_copy["timestamp"] = \
    twitter_archive_data_copy["timestamp"].apply(pd.Timestamp)

twitter_archive_data_copy["retweeted_status_timestamp"] = \
    twitter_archive_data_copy["retweeted_status_timestamp"].\
        apply(pd.Timestamp)
```

- `source` 中保存了 `HTML` 标签类型数据

  > 对该 `field` 的数据独特值进行探索，发现了该 `field` 目前主要包括了 `4` 种类型的值。该 `field` 的问题主要是以 `HTML` 标签的形式来储存数据，它需要被转换为合理的数据。

```
twitter_archive_data_copy["source"] = \
    twitter_archive_data_copy["source"].apply(html_content)

twitter_archive_data_copy["source"] = \
    twitter_archive_data_copy["source"].map(source_dict)
```

  > 上面 `html_content` 为需要获取 `tag` 的内容的函数。从官方文档可以指导这几个值是表示发送信息时的设备，所以通过 `source_dict` 进行映射调整数据值以优化数据。

- `text` 中保存了评分数据，`tweet` 文本信息以及链接数据以及使用空字符串保存缺失值

  > 该 `field` 的数据内容复杂性，因此需要从正则表达式的发那个方式去解决该问题，重点是怎么建立一个有效的 `regx pattern`

```
pattern = re.compile("\s{0,1}(\d{1,})/(\d{0,3}).{0,}|\s{.*\w\s{0,1}(https:/
```

> 经过直接统计计算出 `tweet` 中提取出的数据和原有的评分数据进行比较，发现两者数据相符。在后续中将保留一个

- `expanded_urls` 存储了重复数据值以及数据值不符合官方文档解释的值

> 根据官方文档的解释，该 `url` 值应该是指向的特定的 `tweet` 链接，因此利用已有的可确认信息将构建一个列表，重新复制给该 `field`

- `field` 拆分为了多个，例如 `doggo`，`floofer`，`pupper`，`puppo`

> 该 `fields` 的处理方式模仿 `expanded_urls` 的方式，为了避免可能被判定为多种类型的狗，所以先将数据按照字符串连接的处理方式组合成新数据，之后使用 `replace` 方法进行替换

```
dog_type = []
for doggo, floofer, pupper, puppo in zip(twitter_archive_data_copy["doggo"]
                                         twitter_archive_data_copy["floofer
                                         twitter_archive_data_copy["pupper'
                                         twitter_archive_data_copy["puppo"]
    dog_type.append(doggo +  floofer + pupper + puppo)

twitter_archive_data_copy["dog_type"] = pd.Series(dog_type)

twitter_archive_data_copy["dog_type"] = \
    twitter_archive_data_copy["dog_type"].apply(lambda x: x.replace("None",
```

3. `tweet_data` 的数据集

- `id` 是 `int64` 的整数类型，实际应当保存为 `object` 类型

> 该问题的解决直接通过 `astype` 的方法对数据类型进行了转换

- `created_at` 是 `object` 的对象数据(字符串模式)，实际应当保存为 `datetime` 类型

> 该 `field` 的转换也是通过 `apply` 的方法分别对各个 `field` 进行了转换。另外需要注意该方法中传入的方法是使用的 `pd.Timestamp` 方法

- `full_text` 中保存了评分数据， `tweet` 文本信息以及链接数据

> 该 `field` 的数据内容复杂性，因此需要从正则表达式的发那个方式去解决该问题，重点是怎么建立一个有效的 `regx pattern`
> 后续需要处理的问题是数据聚合的时候比较得分数据之间是否存在差异

- `media_url`，`in_reply_to_status_id` 以及 `in_reply_to_user_id` 中以 `None` 来保存缺失数据的情况

> 该 `field` 的数据内容通过 `apply` 的方法，利用 `lambda` 创建了一个条件语句 将数据

> `None` 数据转换为合适的缺失值

```
tweet_data_copy[["media_url", "in_reply_to_status_id", "in_reply_to_user_id
    tweet_data_copy[["media_url", "in_reply_to_status_id", "in_reply_to_use
        applymap(lambda x: np.NaN if x==None else x)
```

- `source` 中保存了 `HTML` 标签类型数据

> 对该 `field` 的数据独特值进行探索，发现了该 `field` 目前主要包括了 `4` 种类型的值。该 `field` 的问题主要是以 `HTML` 标签的形式来储存数据，它需要被转换为合理的数据。

```
tweet_data_copy["source"] = \
    tweet_data_copy["source"].apply(html_content)

tweet_data_copy["source"] = \
    tweet_data_copy["source"].map(source_dict)
```

> 上面 `html_content` 为需要获取 `tag` 的内容的函数。从官方文档可以指导这几个值是表示发送信息时的设备，所以通过 `source_dict` 进行映射调整数据值以优化数据。

融合数据集的后处理

考虑到融合数据需要依照 `outer joint` 的方式进行，因此优先考虑对大数据量的数据进行融合——这样保证了数据一致性。因此分别通过两步进行了融合。融合后目前得到的数据为有 `2356` 个数据点且包含了 `36` 个 `field` 数据。但是数据依然存在以下问题，需要进行相关处理：

- 对融合后的数据集进行最后探索，存在某些 `field` 的值缺失严重，无法通过相应的信息进行转化。因此对该类型的 `field` 进行 `drop` 处理。
- 某些 `field` 之间存在重复保存值，进行缺失化处理，以避免后期数据出现重复使用的情况

清理之后，最终得到的数据为 `2356` 个数据点且包含了 `24` 个 `field` 数据