关于价格更新 是分组更新的

1、

价格处理方式：基于 抓取的“Amazon asin(或者product_id)当前的价格数据”，参考 shopify 旧价格数据，计算出 null_zero_data, noraml_data, minus_data （这里的 null, zero 都是对于 product_id 当前的价格数据 而言的，但又是基于旧价格也是null（或者没有抓取到）排序的）

参考代码 ***:/remote_download/price_split_helper.py(def split_price) && /config/configuration.py(def price_special_case_manager)***

price_split_helper.py(def split_price)

```
    # no price info
    zero_price = price_detail[price_detail.loc[:, 'basic_price'] == 0]
    null_price = price_detail.loc[price_detail['basic_price'].isnull(), :]
    null_zero_list = pd.merge(zero_price, null_price, how="outer")

    # normal price data
    price_detail = price_detail.dropna(subset=['basic_price'])
    price_detail.loc[:, 'new_price'] = pd.Series((get_sale_price(store_name, x)
                                                  for x in price_detail['basic_price']), index=price_detail.index)

    # old price info
    reference_price = reference.loc[:, ["product_id", "variant_id", "old_price", "quantity"]]
    merged_null_zero_price = pd.merge(null_zero_list, reference_price, how="left", on=["product_id", "variant_id"])
    merged_price_detail = pd.merge(price_detail, reference_price, how="left",
                                   on=["product_id", "variant_id"])

    # deal with old price missing data - no inventory info
    merged_null_zero_price = price_special_case_manager(merged_null_zero_price, "OldPriceMissing")
    init_size = merged_null_zero_price.shape[0]
    merged_price_detail = price_special_case_manager(merged_price_detail, "OldPriceMissing", init_size)
```

configuration.py(def price_special_case_manager)

```
if case == "OldPriceMissing":
    null_data_list = merged_data.loc[merged_data["old_price"].isnull(), :]
    normal_data_list = merged_data.dropna(subset=["old_price"])
    norma_size = normal_data_list.shape[0]
    if norma_size >= limit_size:
        mix_merged_data = normal_data_list
    else:
        null_data_size = limit_size - norma_size
        null_data_list.fillna({"old_price": pd.np.nan, "quantity": 3}, inplace=True)
        null_data_list = null_data_list.sort_values("basic_price")
        null_data_list = null_data_list.head(null_data_size)
        mix_merged_data = pd.merge(null_data_list, normal_data_list, how="outer")
```

2、

更新顺序：null -> 0 -> normal（灰条区，会根据价格区间差，再次分组，计算优先更新的） -> minus (价格变动了，但是比以前的价格低了，由于"一天最大更新量"的限制 一般不会更新到这个)

参考代码 ***/config/configuration.py(def split_price_by_value***）

```
def split_price_by_value(price_data, filter_name):
    columns = price_data.columns

    if "sort_value" in columns:
        plus_price_data = price_data[price_data["sort_value"] > 0]
        minus_price_data = price_data[price_data["sort_value"] < 0]
    else:
        plus_price_data = price_data
        minus_price_data = pd.DataFrame(columns=columns)
    abnormal_inventory = price_data.query("sort_value == 0 & quantity == 0")
    group_null = plus_price_data.loc[plus_price_data[filter_name].isnull(), :]
    plus_price_data = plus_price_data.dropna(subset=[filter_name])
    group_zero = plus_price_data[plus_price_data[filter_name] == 0]
    group_one = plus_price_data.query("{} > 0 & {} <= 100".format(filter_name, filter_name))
    group_two = plus_price_data.query("{} > 100 & {} <= 200".format(filter_name, filter_name))
    group_three = plus_price_data[plus_price_data[filter_name] > 200]
    groups = [
        (-1, group_null),
        (0, abnormal_inventory),
        (1, group_zero),
        (2, group_one),
        (3, group_two),
        (4, group_three),
        (5, minus_price_data)
    ]
    # print(group_one.sort_values("sort_value", ascending=False).head(5000))
    # print(group_one.head(-100))
    return groups
```

默认一天最大更新量：50000（shopify 有限制，另外防止程序跑的时间 >24h, 影响第二天的更新）

参考代码：***/config/configuration.py(def def price_special_case_manager)***

```
def price_special_case_manager(merged_data, case=None, init_size=0):
    default_max_size = 50000
```

3、
具体执行价格分类更新的代码：

/remote_download/shopify_price_updater.py(def product_variant_price_update_by_pandas_data)

```
    def product_variant_price_update_by_pandas_data(self, price_data_info):
        # default columns = ["product_id", "variant_id", "basic_price", "sort_value"]
        product_price_data = price_data_info
        columns = list(product_price_data.columns)
        variant_id_index = self._columns_index_check(columns, "variant_id", "variantid", "variant id")
        price_index = self._columns_index_check(columns, "basic_price", "price", "basic price")
        product_id_index = self._columns_index_check(columns, "product_id", "productid", "product id")
        product_id_name = columns[product_id_index]
        variant_id_name = columns[variant_id_index]
        price_name = columns[price_index]
        product_price_data[product_id_name] = product_price_data[product_id_name].astype(str)
        product_price_data[variant_id_name] = product_price_data[variant_id_name].astype(str)
        groups = split_price_by_value(product_price_data, price_name)

        group_start, index_start = self._init_update_log_record()
        for group in groups:
            group_num, price_data = group
            print("group- {}  size- {}".format(group_num, price_data.shape[0]))
            if group_num >= group_start:
                if group_num > group_start:
                    index_start = 0
                    self._init_update_log_record(group_num, 0, 0, price_data.shape[0])
                if not price_data.empty:
                    if "sort_value" in columns:
                        price_data = price_data.sort_values("sort_value", ascending=False)
                    else:
                        price_data = price_data.sort_values(price_name)
                    product_id_list = price_data[product_id_name].unique().tolist()
                    price_data.set_index(product_id_name, drop=False, inplace=True)
                    size = len(product_id_list)

                    merchant_info_list = []
                    for product_id in product_id_list:
                        index = product_id_list.index(product_id)
                        try:
                            if index >= index_start:
                                print(product_id)
                                product = price_data.loc[product_id]
                                try:
```

/config/configuration.py(def split_price_by_value）

```python
def split_price_by_value(price_data, filter_name):
    columns = price_data.columns

    if "sort_value" in columns:
        plus_price_data = price_data[price_data["sort_value"] > 0]
        minus_price_data = price_data[price_data["sort_value"] < 0]
    else:
        plus_price_data = price_data
        minus_price_data = pd.DataFrame(columns=columns)
    abnormal_inventory = price_data.query("sort_value == 0 & quantity == 0")
    group_null = plus_price_data.loc[plus_price_data[filter_name].isnull(), :]
    plus_price_data = plus_price_data.dropna(subset=[filter_name])
    group_zero = plus_price_data[plus_price_data[filter_name] == 0]
    group_one = plus_price_data.query("{} > 0 & {} <= 100".format(filter_name, filter_name))
    group_two = plus_price_data.query("{} > 100 & {} <= 200".format(filter_name, filter_name))
    group_three = plus_price_data[plus_price_data[filter_name] > 200]
    groups = [
        (-1, group_null),
        (0, abnormal_inventory),
        (1, group_zero),
        (2, group_one),
        (3, group_two),
        (4, group_three),
        (5, minus_price_data)
    ]
    # print(group_one.sort_values("sort_value", ascending=False).head(5000))
    # print(group_one.head(-100))
    return groups
```

/remote_download/shopify_price_updater.py(def _init_update_log_record)
(log 文件的输出程序，也是 获取更新价格组 *group_start(*貌似默认***0-abnormal_inventory*** 价格组*)*，也是防止程序意外中断，不会从头更新，而是从上一次最近一次记录（这个与*google mecharnt* 数据更新有关）*)*

```python
def _init_update_log_record(self, group=None, start_id=None, id_index=None, size=None, note=None, update=False):
    if group or start_id or size:
        time_stamp = time.strftime("%Y-%m-%d %H:%M:%S", time.localtime())
        if not note:
            note = "Processing"
            if id_index == size:
                note = "Done"
        data = [group, start_id, id_index, size, note, time_stamp]
    else:
        data = None
    log_record_file = os.path.join(self.log_file_dir, self.log_save_name)
    log_columns = ["group", "id", "index", "size", "note", "time"]
    if os.path.exists(log_record_file):
        log_data = pandas.read_csv(log_record_file, delimiter="\t")
    else:
        log_data = pandas.DataFrame(columns=log_columns)

    if not data:
        if log_data.empty:
            group_start = 0
            index_start = 0
        else:
            log_size = log_data.shape[0]
            last_group_start = log_data["group"].tolist()[-1]
            last_id_start = log_data["id"].tolist()[-1]
            last_note = log_data["note"].tolist()[-1]

            if last_note == "Done":
                processing_log = log_data[log_data.loc[:, "note"] == "Done"]
            else:
                processing_log = log_data[log_data.loc[:, "note"] == "Processing"]
                if processing_log.empty:
                    processing_log = log_data[log_data.loc[:, "note"] == "Done"]
```

4、
具体执行shopify价格更新的代码：

***/config/configuration.py*（*def get_sale_price*）**

```python
def get_sale_price(shop_name_abbr, basic_price):
    if basic_price != basic_price:
        sale_price = None
    else:
        try:
            basic_price = float(basic_price)
        except ValueError:
            basic_price = None
        except TypeError:
            basic_price = None

        if basic_price:
            if shop_name_abbr in ["ST", "TS", "OL"]:
                price1 = 1.6 * basic_price
                price2 = basic_price + 14.0
                sale_price = round(max(price1, price2), 2)

            else:
                price1 = 1.5 * basic_price
                price2 = basic_price + 12.0
                sale_price = round(max(price1, price2), 2)
        else:
            sale_price = basic_price
    return sale_price
```

***/remote_download/shopify_price_updater.py*（*def _variant_price_update*）**

```python
def _variant_price_update(self, variant_id, basic_price):
    new_sale_price = get_sale_price(self.shop_name_abbr, basic_price)
    if basic_price != basic_price:
        basic_price_show = "Nan"
    else:
        if basic_price:
            basic_price_show = basic_price
        else:
            basic_price_show = "None"
    if new_sale_price:
        new_sale_price_show = new_sale_price
    else:
        new_sale_price_show = False
    try:
        variant = shopify.Variant.find(variant_id)
    except Exception as e:
        error_info = str(e)
        print(error_info)
        return None
    else:
        if new_sale_price:
            price_status = True
            new_compare_price = round(new_sale_price * 1.2 + random.uniform(1, 5), 2)
            old_sale_price = float(variant.price)
            if new_sale_price != old_sale_price:
                variant.price = new_sale_price
                variant.compare_at_price = new_compare_price
                if not variant.inventory_quantity:
                    variant.inventory_quantity = 3
                save_status = self._save_product_attribute(variant)
            else:
                save_status = True
        else:
            if basic_price_show == "None":
                price_status = False
                old_sale_price = float(variant.price)
                # variant.price = 0
                variant.inventory_quantity = 0
                save_status = self._save_product_attribute(variant)
            else:
                save_status = False
```