FastFasion

Case Study

Problem Statement 1

Aggregate the Quantity for each Store-Style level at a week level (week starts from Monday) and identify the top products being sold at each of the stores. Highlight if there is any change in product preference in 2019 as compared to 2018.

The questions can be broken down into 3 parts :

1. Aggregate the quantity sold or each Store-Style level at a week level
    - Week starts from Monday
    - Quantity sold -> transactions
2. Identify the top products being sold at each of the stores.
3. Highlight if there is any change in product preference in 2019 as compared to 2018.

Solution 1

STEPS For Part 1 :

1. Find the iso_week of the transactions from the transactions table
2. Merge the transaction data with the master product data
3. Group by store_id, iso_week, and stylecode_id to get the Store-Style level at week level data
4. The Result can be seen in Figure 1.1

Answering Q1

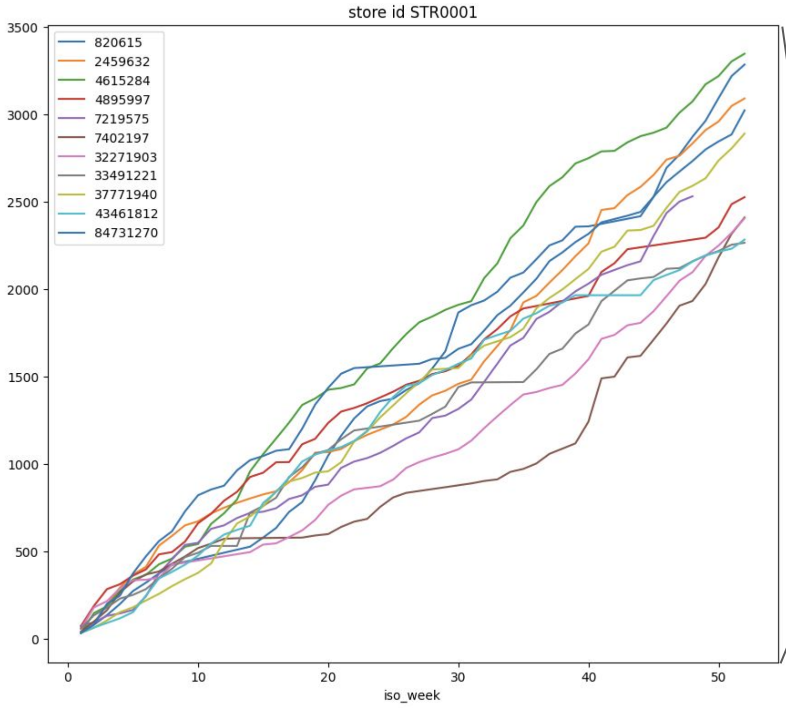
Show code

	store_id	stylecolor_id	iso_week	qty
0	STR0001	32284	1	45.000
1	STR0001	32284	2	13.000
2	STR0001	32284	3	18.000
3	STR0001	32284	4	0.000
4	STR0001	32284	5	11.000
...	...	...	...	...
127983	STR0010	89741435	48	47.000
127984	STR0010	89741435	49	2.000
127985	STR0010	89741435	50	30.000
127986	STR0010	89741435	51	7.000
127987	STR0010	89741435	52	15.000

127988 rows × 4 columns

Figure 1.1 :
Table of Store-Style level at week level

Solution 1



1. These are 11 styles that were top sellers across all the stores :
   820615, 4615284, 4848421, 5075479, 7949461, 8407512, 8785174, 29101804, 32271903, 33491221, 65121222

2. There is huge variation is total sales between the best of the top sellers in a store, verses the worst of the top sellers

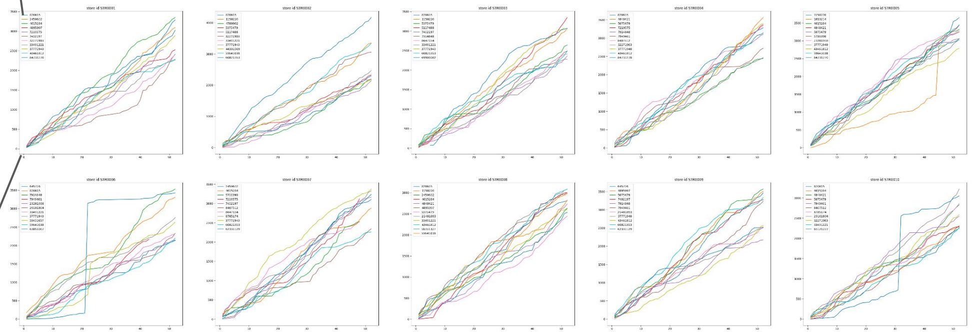


Figure 1.3 : Plot of cumulative weekly sales for top selling styles

Solution 1

STEPS For Part 3 :

1. Data for 2018 was missing in the transaction table, while the data for 2019 was made available. Thus, a comparison study could not be made between the sales data for 2018 and 2019

2. Provided the data the similar approach can be taken to compare the top styles being sold between the two years

Problem Statement 2

Detect the Outliers in the Quantity for each Store-Product combination and apply an outlier treatment on the same. Specify the outlier treatment technique

STEPS for Solution :

1. Detect outliers in residuals, post trend, seasonal decomposition
2. Test Different approaches
    - IQR
    - Z-score
    - Exponential Moving Average

Solution 2

Point Outliers / Spike Outliers :

1. Based on visualization we can observe that the outlier types are spikes (Figure 2.1)

2. Outlier can be seen in residual plots - made post decomposition (Figure 2.2) - challenge being estimating seasonality time period. Thus ema was used (shown below).

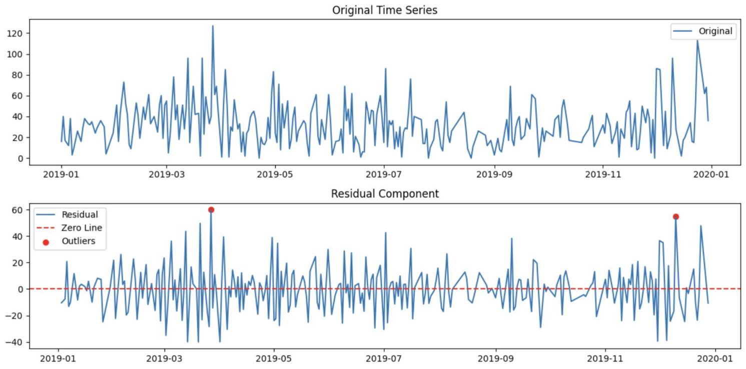


Figure 2.1 : Residual plot with outliers being marked

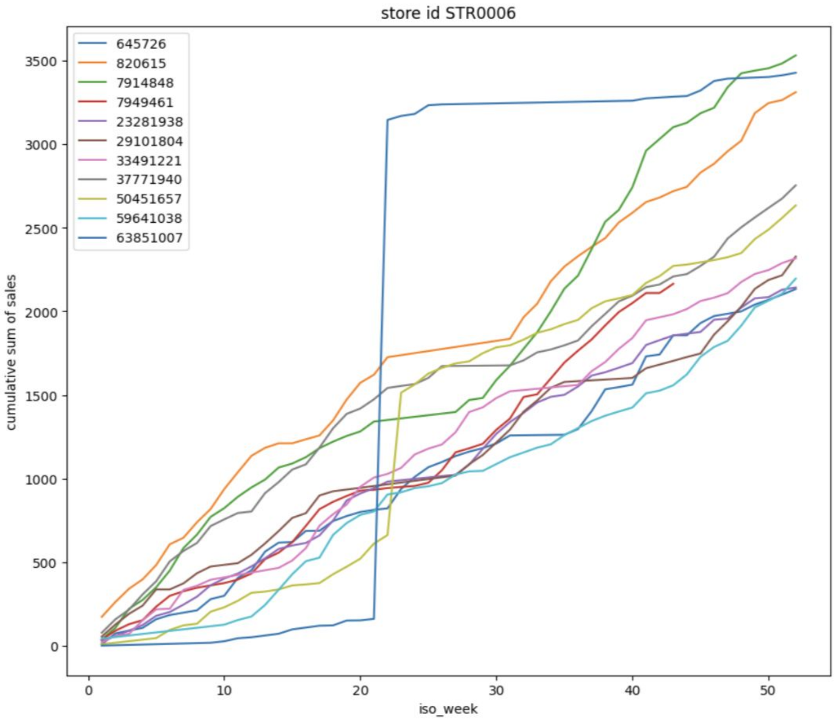


Figure 2.2 : Spike Outliers in Sales data

Solution 2

Outlier removal can be done using the 3 approaches mentioned below. The table also shows a comparison of each of the methods:

Approach	Pros	Cons
IQR (Interquartile Range)	- Robust to outliers - Simple to compute	- Data-driven imputation - May introduce bias
Z-Score (Standard Score)	- Utilizes standardization - Detects and handles outliers	- Data-driven imputation - Loss of information
EMA (Exponential Moving Average)	- Temporal awareness - Smoothing effect	- Sensitive to parameter selection - Does not handle outliers - Doesn't provide statistical boundaries

- Use IQR when robustness to outliers is crucial, and you are not concerned with capturing temporal patterns
- Use Z-score when you want to standardize the data and handle outliers effectively, but are not focused on temporal aspects
- Use EMA when you want to impute missing values while considering the time series' temporal structure, especially for capturing trends and seasonality. However, be cautious with parameter selection and outliers

Ref : https://s-ai-f.github.io/Time-Series/outlier-detection-in-time-series.html#introduction-1

Solution 2

All three outlier removal methods are compared for the % of transactions that result to being tagged as outlier :

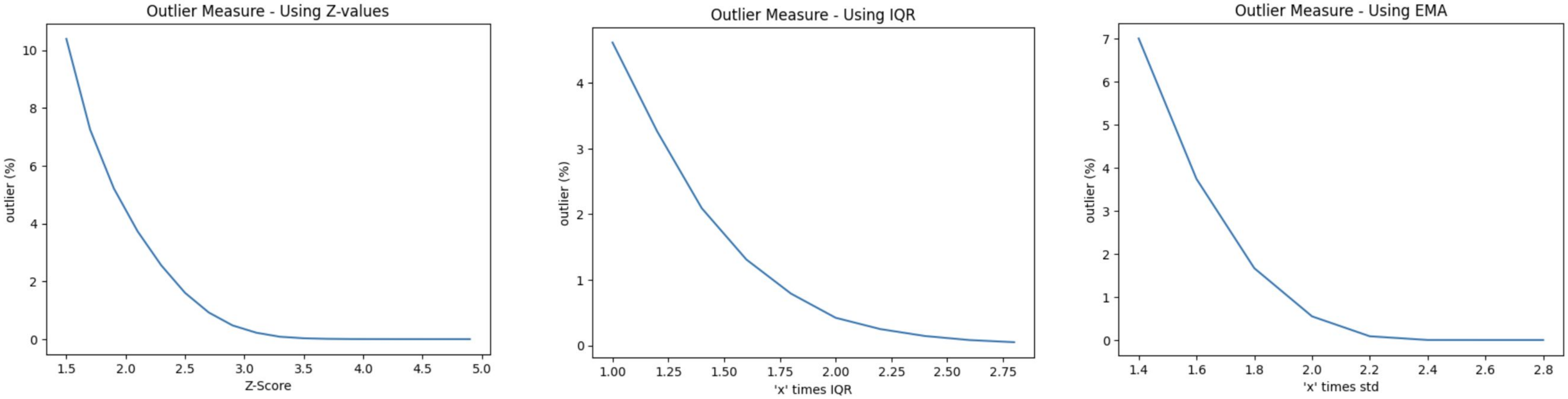


Figure 2.3 : Plot of cumulative weekly sales for top seller products

We choose Exponential Moving Average approach as it incorporates trend and is time aware, which makes it a more robust technique than Z-score or IQR.

If the residual of a point was 1.8 times the standard deviation of all residuals, it was considered an outlier. Using this technique only 1.8% of the transactions were recorded as outliers.Such data points were removed from the data set.

Problem Statement 3

Check the data for missing values at Store-Product Level and apply the missing value treatment by imputing the missing values by either mean or median or any other value. Also provide the reasoning for the missing value treatment method used

The questions can be broken down into 3 parts :

- Check the data for missing values at Store-Product Level and
- Apply the missing value treatment by imputing the missing values
- Provide the reasoning for the missing value treatment method used

Solution 3

There were two columns with null values observed :

- **Date Column -**
  For the date columns, an algorithm was developed to detect the missing date from the sales data. and corresponding date was selected.
  This algorithm can be improved

- **Quantity Column -**
  For missing quantity, the EMA was used impute the sale quantity. EMA was used because it captures trend as well as time series component.

```
[64] txns[txns.date.isnull()]
```

	date	store_id	sku_id	qty	iso_week	mean_sp	std_sp	lowerq_sp	upperq_sp	Z_score	IQR	ema	residual	outlier
22359	NaT	STR0001	369066292968	1.000	<NA>	7.529	7.058	2.000	11.750	-0.925	9.750	7.825	-6.825	0.000
42198	NaT	STR0002	306149104992	2.000	<NA>	7.950	6.920	2.000	13.000	-0.860	11.000	4.305	-2.305	0.000
42199	NaT	STR0002	306149104992	0.000	<NA>	7.950	6.920	2.000	13.000	-1.149	11.000	19.435	-19.435	0.000
80429	NaT	STR0003	304414346297	5.000	<NA>	14.733	8.675	8.000	21.750	-1.122	13.750	10.805	-5.805	0.000
331499	NaT	STR0009	476267354270	1.000	<NA>	8.157	6.792	2.000	13.000	-1.054	11.000	14.638	-13.638	0.000
337749	NaT	STR0009	304493977342	9.000	<NA>	7.243	6.555	1.250	11.750	0.268	10.500	3.555	5.445	0.000

```
txns[txns.qty.isnull()]
```

	date	store_id	sku_id	qty	iso_week	mean_sp	std_sp	lowerq_sp	upperq_sp	Z_score	IQR	ema	residual	outlie
17163	2019-06-08	STR0001	476252256352	NaN	23	8.228	6.756	3.000	11.000	NaN	8.000	11.185	NaN	0.00
17643	2019-02-12	STR0001	492195493980	NaN	7	7.056	6.150	2.000	11.000	NaN	9.000	16.545	NaN	0.00
152024	2019-05-12	STR0004	369058443710	NaN	19	8.180	7.376	2.000	14.000	NaN	12.000	9.395	NaN	0.00
179968	2019-06-26	STR0005	304451963502	NaN	26	7.620	6.624	2.000	12.000	NaN	10.000	13.218	NaN	0.00
282463	2019-08-11	STR0008	304498237391	NaN	32	8.265	7.499	3.000	11.000	NaN	8.000	7.186	NaN	0.00
286628	2019-01-25	STR0008	315670498865	NaN	4	7.875	6.483	2.750	12.250	NaN	9.500	1.975	NaN	0.00
286646	2019-04-23	STR0008	315670498865	NaN	17	7.875	6.483	2.750	12.250	NaN	9.500	10.606	NaN	0.00
288365	2019-05-12	STR0008	498088294215	NaN	19	7.435	6.344	2.000	11.000	NaN	9.000	11.704	NaN	0.00
339733	2019-03-08	STR0009	490251133500	NaN	10	8.118	7.506	2.000	12.000	NaN	10.000	5.915	NaN	0.00
339747	2019-05-20	STR0009	490251133500	NaN	21	8.118	7.506	2.000	12.000	NaN	10.000	0.983	NaN	0.00
369063	2019-10-09	STR0010	498099958319	NaN	41	7.246	5.910	2.000	11.000	NaN	9.000	16.768	NaN	0.00

Solution 3

imputing - dates

```
def find_missing_date(store_id = 'STR0009',sku_id = 304493977342) :

    temp = txns[
        (txns.store_id == store_id) &
        (txns.sku_id == sku_id)
    ]
    temp['days'] = (temp.date - temp.date.min()).dt.days

    temp['checker_col'] = 1
    temp['checker_col'] = temp['checker_col'].cumsum()-1

    temp['checker'] = temp['checker_col'] == temp['days']
    first_false_index = temp['checker'].index[~temp['checker']].tolist()[0]

    # print(first_false_index)
    # display(temp)


    days = temp.loc[first_false_index,'days'] -1


    return temp.date.min() + pd.Timedelta(days=days)
```

```
[69] find_missing_date()

     Timestamp('2019-02-02 00:00:00')
```

```
[70] txns.loc[txns.date.isnull(),'date'] = txns[txns.date.isnull()].apply(lambda i : find_missing_date(i.store_id,i.sku_id),axis=1)
```

```
[71] txns.loc[txns.iso_week.isnull(),'iso_week'] = txns.loc[txns.iso_week.isnull(),'date'].dt.isocalendar().week
```

- **Date Column -** For the date columns, an algorithm was developed to detect the missing date from the sales data and corresponding date was selected. The algorithm can be seen in Figure on the left.

  Why? - Later Investigation revealed that the sales is not continuous. And thus this algorithm was not appropriate. But since the missing count was low, and for the interest of time, this wasn't prioritized

- **Quantity Column -** For missing quantity, the EMA was used to impute the sale quantity. It can be seen on the right.

  Why? - It captures trend of the time series component.

```
[73] def ema_imputer(missing_index,temp=txns):

         temp['ema'] = temp.groupby(['store_id','sku_id']).qty.ewm(span=2, adjust=False).mean().reset_index()['qty']
         return temp.loc[temp.index.isin(missing_index),'ema']
```

```
[74] missing = txns[txns.qty.isnull()]
     missing_index = missing.index
```

```
[75] txns.loc[txns.qty.isnull(),'qty'] = ema_imputer(missing_index,txns)
```

Problem Statement 4

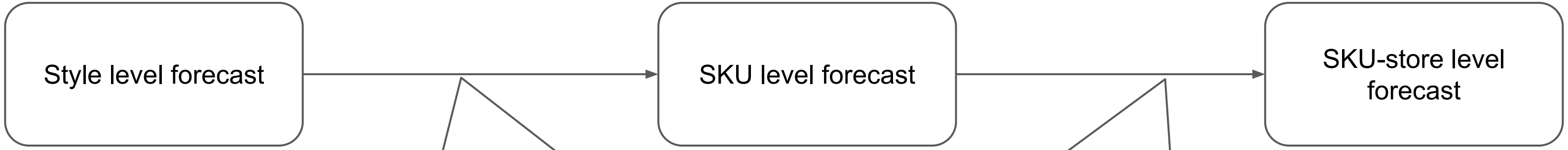
Split the future forecast given at Style-Week level into Store-SKU-Week level using appropriate logic. Please specify the logic and rationale behind the decision. Also, highlight the approach to handle new products.

The questions can be broken down into 3 parts :

- Split the future forecast given at Style-Week level into Store-SKU-Week level using appropriate logic
- Please specify the logic and rationale behind the decision
- Highlight the approach to handle new products

Solution 4

Solution Flow diagram



- Clustering is used to group styles that have similar size ratio.
- This clustering can future be augmented with, data like material, color, and other characteristics to predict the the split across new products.

- Clustering is used to group styles that have similar store-distribution ratio.
- This clustering can future be augmented with, data like material, color, and other characteristics to predict the the split across new products.

Solution 4

Explored how to break down from styles to SKU. It was observed that for each Style there were different sizes, which map to different SKU

Therefore we had to map each style into the different sizes. Since, Historical transaction can be mapped into the forecast, we explored the transaction data. Using the ratio of transactions done across different sizes of each style, we created a map. This can be seen in Figure 4.1. For each of the 430 styles there were different ratios.

These styles were then grouped into 8 groups. Using clustering. Each group having unique ratios of sizes. This can be observed in the Figure 4.2. The grouping was done using K-means clustering.

Figure 4.1 :
Style to Size-Ratio

size	stylecolor_id	L	M	S
0	32284	0.374	0.447	0.179
1	56354	0.290	0.403	0.306
2	164052	0.378	0.291	0.330
3	185375	0.000	0.000	1.000
4	268743	1.000	0.000	0.000
...	...	...	...	...
425	87721564	0.000	0.593	0.407
426	88061934	1.000	0.000	0.000
427	88981853	0.366	0.304	0.329
428	89521319	0.000	0.534	0.466
429	89741435	0.272	0.289	0.439

Figure 4.2 :
Style-Cluster to Size-Ratio

size	cluster_no	L	M	S
0	0	0.370	0.337	0.291
1	1	0.000	0.000	1.000
2	2	0.000	0.511	0.481
3	3	1.000	0.000	0.000
4	4	0.000	1.000	0.000
5	5	0.503	0.497	0.000
6	6	0.517	0.000	0.483
7	7	0.286	0.299	0.403

Solution 4

We can observe that each cluster size distribution is unimodal. Making the clustering impactful, and distinctive from each other. This grouping was effective (Figure 4.3)

In Figure 4.4 you can observe the elbow plot of the clusters

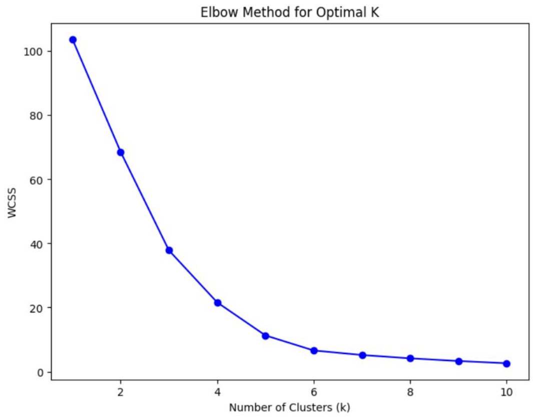


Figure 4.4 : Clustering Elbow Plot

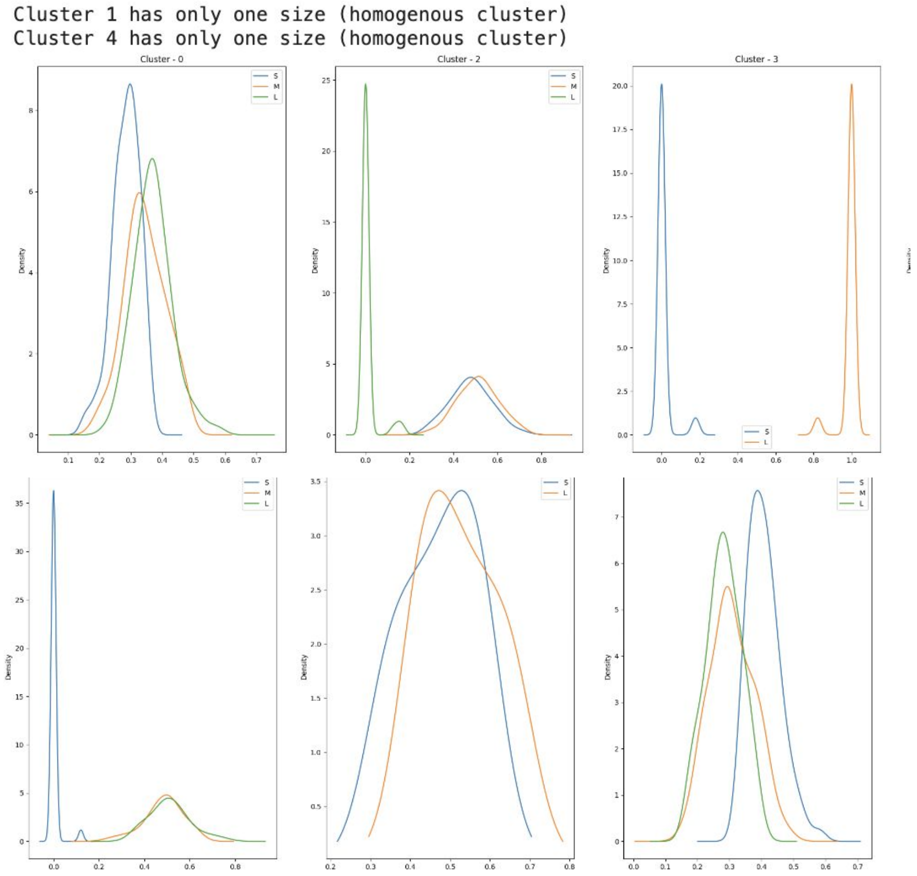


Figure 4.3 : Style-Cluster to Size-Ratio

Solution 4

We had to find a mapping for each style into different stores. Since transaction data can be used to split the forecast. we explored the store distribution in the transaction data. Using the ratio of transactions done across different stores of each style, we created a map.

These styles were then grouped into 16 groups. Each group having unique ratios store distribution. This can be observed in the Figure 4.5. The grouping was done using K-means clustering.

This grouping was effective as it can be observed that each of the cluster has unimodal distribution Figure 4.5

Figure 4.5 : Style-Cluster to Store Rato

store_id	cluster_no	STR0001	STR0002	STR0003	STR0004	STR0005	STR0006	STR0007	STR0008	STR0009	STR0010
0	0	0.121	0.070	0.064	0.041	0.111	0.112	0.086	0.164	0.072	0.162
1	1	0.136	0.073	0.109	0.085	0.140	0.134	0.086	0.105	0.090	0.063
2	2	0.071	0.090	0.068	0.155	0.112	0.113	0.156	0.102	0.051	0.079
3	3	0.112	0.083	0.126	0.146	0.038	0.144	0.033	0.101	0.066	0.114
4	4	0.121	0.149	0.049	0.063	0.095	0.056	0.138	0.139	0.107	0.064
5	5	0.129	0.130	0.155	0.062	0.063	0.094	0.167	0.067	0.048	0.080
6	6	0.068	0.113	0.134	0.101	0.105	0.096	0.073	0.126	0.132	0.072
7	7	0.035	0.137	0.109	0.060	0.094	0.143	0.122	0.045	0.120	0.142
8	8	0.107	0.086	0.061	0.139	0.085	0.080	0.102	0.073	0.137	0.130
9	9	0.132	0.133	0.085	0.157	0.181	0.000	0.000	0.060	0.107	0.049
10	10	0.000	0.000	0.289	0.000	0.247	0.000	0.000	0.000	0.271	0.000
11	11	0.109	0.000	0.153	0.116	0.135	0.000	0.092	0.114	0.073	0.139
12	12	0.085	0.134	0.169	0.081	0.000	0.000	0.071	0.071	0.171	0.162
13	13	0.162	0.144	0.000	0.066	0.132	0.130	0.047	0.056	0.146	0.108
14	14	0.000	0.000	0.237	0.000	0.346	0.000	0.000	0.302	0.000	0.000
15	15	0.000	0.197	0.000	0.167	0.000	0.103	0.000	0.216	0.000	0.197

Solution 4

We can observe that each cluster size distribution is mostly unimodal. Making the clustering impactful, and distinctive from each other. This grouping was effective (Figure 4.6)

How was new product handled* :

There were 26 style codes that were missing in the transaction data in 2019. The assumption is that these are forecasted sales based on multiple years of data. While there have been no sales for these styles in the last year. Thus, this was a mistake by the forecasting team and these were forecasted to 0.

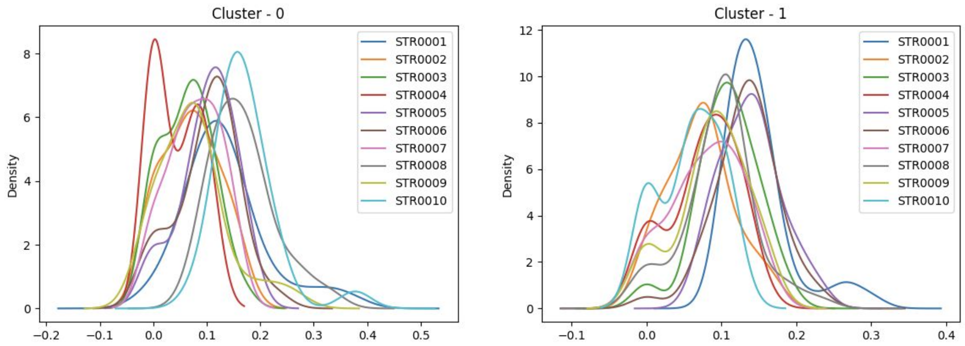


Figure 4.5 : Distribution before clustering

```
"""
since clusters are developed using txn data ->
clusters will be missing for the 26 stylecode_id (scid) that
doesnt have any sales : for this we will assume the existing ratio ... ??

My assumption is that the forcasting team should have overserved this
while forecasting / this is a miss by the forecasting team -
Thus, I will be overwriting these forecasts and making them 0
"""

forecast[forecast.cluster_no.isnull()]

forecast.loc[forecast.cluster_no.isnull(),'L'] = 0
forecast.loc[forecast.cluster_no.isnull(),'M'] = 0
forecast.loc[forecast.cluster_no.isnull(),'S'] = 0
```

Figure 4.6 : Distribution before clustering

*While creating the notebook the following approach to handle the missing styles in transaction data was considered. Later while re-looking at the questions it was proposed to look at these style codes as new products. Can explore ideas during discussion call

Problem Statement 5

Given the latest inventory position of stores (as per “Store_inventory” file), generate next 4 allocation plans (one for each of the first 4 weeks of 2020). In scenarios with limited inventory (less than store level demand), the priority should be given to the store having higher sales potential.

The questions can be broken down into 2 parts :

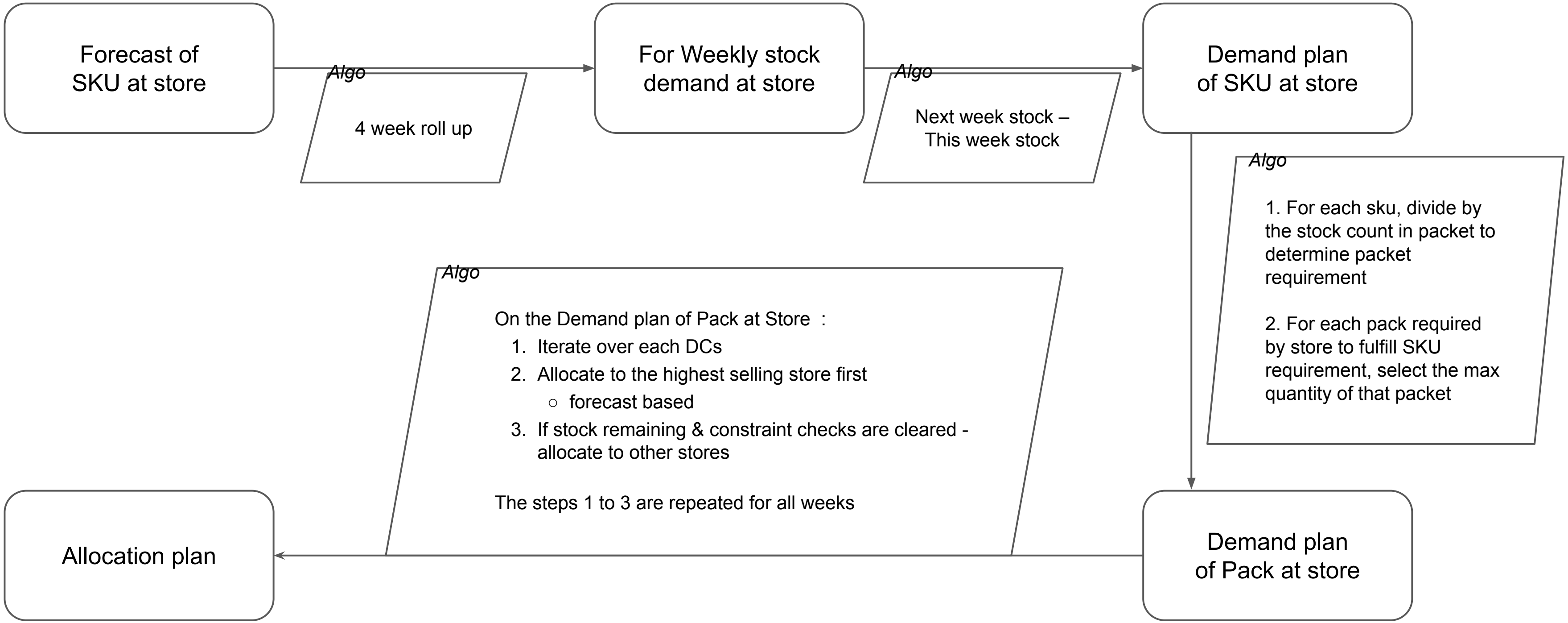
- Given the latest inventory position of stores (as per “Store_inventory” file), generate next 4 allocation plans
- In scenarios with limited inventory (less than store level demand)
  - The priority should be given to the store having higher sales potential.

Other constraints :

- FastFashion wants to maintain a stock to satisfy the next 4 weeks of demand for each SKU and each store at any point in time

Solution 5

Solution Flow diagram



Solution 5

For “Weekly stock demand at store” : First we create a rolling stock level at a 4 week basis. This is the level of stock that needs to be maintained in the beginning of the week. The Figures 5.1 show the code and the stock level requirement :

```
# creating the stock level to be maintained each week
rolling_sum = forecast.groupby(['sku_id'])[stores].rolling(window=4).sum().reset_index()

# removing the weeks there are no demand forecasted
rolling_sum = rolling_sum[~rolling_sum.STR0001.isnull()]

# Function to generate row numbers within each group
def row_number(group_df):
    group_df['week_num'] = range(1, len(group_df)+1)
    return group_df

# Apply the row_number function within each group using groupby
rolling_sum = rolling_sum.groupby('sku_id').apply(row_number)
```

	sku_id	level_1	STR0001	STR0002	STR0003	STR0004	STR0005	STR0006	STR0007	STR0008	STR0009	STR0010	week_num
3	304400035889.000	6747	48.000	77.000	105.000	48.000	80.000	101.000	97.000	61.000	143.000	115.000	1
4	304400035889.000	6748	51.000	82.000	110.000	51.000	84.000	107.000	102.000	64.000	151.000	121.000	2
5	304400035889.000	6749	51.000	81.000	110.000	51.000	84.000	107.000	102.000	64.000	151.000	121.000	3
6	304400035889.000	6750	46.000	74.000	100.000	46.000	76.000	97.000	93.000	58.000	137.000	109.000	4
7	304400035889.000	6751	46.000	74.000	100.000	46.000	75.000	97.000	93.000	58.000	137.000	109.000	5
...	...	...	...	...	...	...	...	...	...	...	...	...	...
8051	498099958319.000	451	11.000	23.000	11.000	28.000	23.000	12.000	29.000	23.000	17.000	18.000	1
8052	498099958319.000	452	10.000	22.000	10.000	27.000	22.000	12.000	28.000	22.000	16.000	17.000	2
8053	498099958319.000	453	10.000	21.000	10.000	26.000	22.000	12.000	27.000	21.000	15.000	17.000	3
8054	498099958319.000	454	9.000	19.000	9.000	24.000	20.000	11.000	25.000	19.000	13.000	16.000	4
8055	498099958319.000	455	5.000	11.000	5.000	14.000	12.000	6.000	14.000	11.000	7.000	9.000	5

Figure 5.1 : Generation Code & “Weekly stock demand at store”

Solution 5

My assumption is that this week's W0 is before W1 of 2020

Stock for January (4 week rolling demand) forecast is shipped before W1 begins - because this is done the min stock is maintained by default, and ignored during allocation plan

I am assuming the code is run/reviewed on Fridays (just for nomenclature purposes) - next_week / this_week

On order stock were ignored for this calculation. They can be directed added post the allocation plan

The function in Figure 5.2 was used to generate “Demand plan of SKU at store” (from flow diagram)

Figure 5.2 : Code for generating Demand plan of SKU at store

```
def df_demand():

  weekly_inv = pd.DataFrame()
  printer = False
  printer2 = False

  for i in range(0,5):

    #Week === i
    # print(i)

    # expected invetory at the begning of the week
    if i==0 :
      #if it is 0th week, on hand stock is my this_weeks_stock_level
      this_week_stock_level = on_hand.copy()
      # display(this_week_stock_level)
      this_week_stock_level.columns = [x.replace("on_hand_","twsl_") if 'on_hand_' in x else x for x in this_week_stock_level.columns]
    else :
      # print(i)
      this_week_stock_level = rolling_sum[rolling_sum.week_num==i]
      this_week_stock_level.columns = ["twsl_"+x if 'STR' in x else x for x in this_week_stock_level.columns]
    if(printer==True) : display(this_week_stock_level)

    # projected sales in the week
    this_week_sales = forecast[forecast.week==i]
    this_week_sales.columns = ["forecast_"+x if 'STR' in x else x for x in this_week_sales.columns]
    if(printer==True) : display(this_week_sales)

    # expected invetory at the begning of next week (tuesday W+1)
    next_week_stock_level = rolling_sum[rolling_sum.week_num==(i+1)]
    next_week_stock_level.columns = ["nwsl_"+x if 'STR' in x else x for x in next_week_stock_level.columns]
    if(printer==True) : display(next_week_stock_level)

    # week_inventory
    week_final = next_week_stock_level.merge(this_week_stock_level,on='sku_id',how='left').merge(this_week_sales,on='sku_id',how='left')
    week_final = week_final.fillna(0)
    if(printer2==True) : display(week_final)

    # demand
    for store in stores:
      week_final['demand_'+store] = week_final['nwsl_'+store] - week_final['twsl_'+store] + week_final['forecast_'+store]

    #weekly_inventory
    week_final['week_num'] = i
    if i == 0 :
      weekly_inv = week_final.copy()
    else :
      weekly_inv = pd.concat([weekly_inv,week_final],axis=0)

  return weekly_inv
```

Solution 5

Demand plan of SKU at store = Next week level – This week stock level
The plans is shared in (Figure 5.3)

	sku_id	week_num	demand_STR0001	demand_STR0002	demand_STR0003	demand_STR0004	demand_STR0005	demand_STR0006	demand_STR0007	demand_STR0008	demand_STR0009	demand_STR0010	pack_id
0	304400035889.000	0	48.000	77.000	105.000	48.000	80.000	101.000	97.000	61.000	143.000	107.000	Pack_368
1	304400035889.000	1	8.000	13.000	17.000	8.000	13.000	17.000	16.000	10.000	24.000	19.000	Pack_368
2	304400035889.000	2	19.000	30.000	41.000	19.000	31.000	40.000	38.000	24.000	56.000	45.000	Pack_368
3	304400035889.000	3	7.000	12.000	16.000	7.000	12.000	15.000	15.000	9.000	22.000	17.000	Pack_368
4	304400035889.000	4	12.000	19.000	26.000	12.000	19.000	25.000	24.000	15.000	35.000	28.000	Pack_368
...	...	...	...	...	...	...	...	...	...	...	...	...	...
5030	498099958319.000	0	5.000	19.000	4.000	22.000	15.000	8.000	24.000	23.000	8.000	11.000	Pack_356
5031	498099958319.000	1	1.000	3.000	1.000	4.000	3.000	2.000	4.000	3.000	2.000	2.000	Pack_356
5032	498099958319.000	2	2.000	3.000	2.000	4.000	4.000	2.000	4.000	3.000	2.000	3.000	Pack_356
5033	498099958319.000	3	1.000	2.000	1.000	3.000	2.000	1.000	3.000	2.000	1.000	2.000	Pack_356
5034	498099958319.000	4	1.000	3.000	1.000	3.000	3.000	1.000	3.000	3.000	2.000	2.000	Pack_356

Figure 5.3 : Demand plan of SKU at store

Solution 5

“Demand plan of Pack at store” from flow diagram (Figure 5.3) :

```
[168] strategy = 'low_stock' #conservative
      strategy2 = 'low_inv' #conservative

      for store in stores:

        if strategy == 'high_stock' :
          demand['pack_demand_'+store] = (demand['demand_'+store]/demand['qty']).apply(lambda x : math.ceil(x))
        if strategy == 'low_stock' :
          demand['pack_demand_'+store] = (demand['demand_'+store]/demand['qty']).apply(lambda x : math.floor(x))
        else :
          print("Wrong Strategy")
          break

      l = ['pack_demand_'+store for store in stores]

      if strategy2 == 'high_inv' :
        pack_demand = demand.groupby(['sku_id','week_num','pack_id'])[l].max().reset_index()
      if strategy2 == 'low_inv' :
        # pack_demand = demand.groupby(['sku_id','week_num','pack_id'])[l].min().reset_index()
        pack_demand = demand.groupby(['week_num','pack_id'])[l].min().reset_index()
      else :
        print("Wrong Strategy")
```

```
pack_demand_store = pack_demand.copy()
pack_demand
```

	week_num	pack_id	pack_demand_STR0001	pack_demand_STR0002	pack_demand_STR0003	pack_demand_STR0004	pack_demand_STR0005	pack_demand_STR0006	pack_demand_STR0007	pack_demand_STR0008	pack_demand_STR0009
0	0	Pack_001	8	2	7	5	9	7	6	7	6
1	0	Pack_002	52	14	46	34	59	46	40	47	40
2	0	Pack_003	0	1	3	2	1	-1	1	2	1
3	0	Pack_004	4	3	0	3	3	4	2	2	4
4	0	Pack_005	14	16	15	12	11	13	10	6	6
...	...	...	...	...	...	...	...	...	...	...	...
2275	4	Pack_452	0	0	0	0	0	0	0	0	0

Figure 5.4 : Generation Code & Demand plan of Pack at store

Solution 5

The Allocation plan required 3 functions :

- **Allocation Algorithm :**
  This function Prioritized the store with higher demand.
  The algo addresses the highest sale potential store first as per the problem statement. (Figure 5.5.1)
- **Check Function :**
  This function checked whether there is available stock in the DC. (Figure 5.5.2)
- **Constraint function :**
  This function validated constraints. (Figure 5.5.3)

```python
print_flag=False
print_flag2=True

dc_inv_temp = dc_inv.copy()
delivery = pd.DataFrame()

for i in sorted(pack_demand_store.week_num.unique()) :

  for j in dc_map.dc_id.unique() :

    store_in_consideration = dc_map[dc_map.dc_id == j].store_id
    store_in_consideration = ['pack_demand_'+store for store in store_in_consideration]
    store_in_consideration.append('pack_id')

    temp = pack_demand_store[pack_demand_store.week_num==i][store_in_consideration]
    if(print_flag) : display(temp)

    temp = temp.melt('pack_id').sort_values(['pack_id','value'],ascending=False)
    if(print_flag) : display(temp)

    temp['variable'] = temp['variable'].apply(lambda x: x.split("_")[-1])
    if(print_flag) : display(temp)

    temp['week_num'] = i
    temp['dc_id'] = j
    if(print_flag) : display(temp)

    temp['possible'] = temp.apply(lambda x: check(x),axis=1)
    if(print_flag) : display(temp)
    if(print_flag2) : display(temp.possible.mean())

    delivery = pd.concat([delivery,temp])
```

Figure 5.5.1 : Allocation Algorithm

```python
def check(x):

  try:
    available_qty = dc_inv_temp[(dc_inv_temp.pack_id==x.pack_id)\
                               & (dc_inv_temp.dc_id==x.dc_id)].qty.values[0]
  except:
    available_qty=0

  if demand_validator(x.variable,x.pack_id,x['value']) :

    if(x['value']<available_qty):

      dc_inv_temp.loc[(dc_inv_temp.pack_id==x.pack_id)\
                      & (dc_inv_temp.dc_id==x.dc_id),'qty'] =\
                                                available_qty - x['value']

      return 1

    else :
      return 0

  else :
    return 0
```

Figure 5.5.2 : DC Inventory Check Algorithm

Figure 5.5.3 : Constraint Check Code

```python
#checking constraints
def demand_validator(store,pack_id,qty):

  skus = pac_config[pac_config.pack_id==pack_id].sku_id
  dept_ids = master[master.sku_id.isin(skus)].dept_id


  # from here
  temp1 = store_inv.merge(master[['sku_id','dept_id']])[['store_id','dept_id']]

  intermediate = pac_config.merge(master[['sku_id','dept_id']])

  try :
    temp2 = delivery[delivery.possible==1].merge(intermediate,on='pack_id')[['variable','dept_id']].rename(
  except :
    return 1

  temp3 = pd.concat([temp1,temp2])

  checker = temp3.groupby(['store_id','dept_id'])[['dept_id']].count().rename(columns={'dept_id':'count_sto

  checker = checker.merge(const,on=['store_id','dept_id'])
  # to here - can be optimized for lesser time

  temp_checker = checker[(checker.store_id==store) & (checker.dept_id.isin(dept_ids))]

  temp_checker.count_store_inv_in_dept_ids = temp_checker.count_store_inv_in_dept_ids+qty

  if (temp_checker.count_store_inv_in_dept_ids > temp_checker['max']).sum() >1:
    return 0
  else:
    return 1
```

Solution 5

Weekly Allocation plan is displayed

Final output Q5 : Allocation Plan -

```python
for i in sorted(delivery.week_num.unique()):
  print("Week %d - Allocation Plan"%i)
  display(delivery[(delivery.week_num==i) & (delivery.possible==1)
  print("\n\n")
```

```
Week 0 - Allocation Plan
```

	dc_id	store_id	pack_id	qty
453	DC0001	STR0001	Pack_454	6
909	DC0001	STR0002	Pack_454	5
1357	DC0001	STR0003	Pack_446	5
441	DC0001	STR0001	Pack_442	5
1809	DC0001	STR0004	Pack_442	5
...	...	...	...	...
919	DC0003	STR0010	Pack_008	4
463	DC0003	STR0009	Pack_008	3
459	DC0003	STR0009	Pack_004	4
915	DC0003	STR0010	Pack_004	3
3	DC0003	STR0008	Pack_004	2

653 rows × 4 columns

Figure 5.6 : W0 - Allocation Plan

Problem Statement 6

Compare the allocation plans with the demand estimate at Store-Week level. Define and calculate a stockout metric to measure effectiveness of each of the allocation plans.

The questions can be broken down into 2 parts :

- Create a dataframe that can be used for comparison
- Create a composite metric

Solution 6

Compare allocation plans with with forecasting or with the rolling demand created? - I am choosing to do it at forecasting level

An algorithm to compare allocation with forecast was made

The Stockout Metric measures - how many times were the demand not met. Every week the demand is not it is counted as a stockout. Same product across different stores are considered as stockout. (Figure 6.2)

There was consideration for stockout to be calculated at DC level, but business context was required to take this decision

This metric can be checked at a store level, or week level granularity.

```
def create_comparison_df(plan) :

  # plan = allocation_plan

  # making transformations to the plan
  plan2 = plan[(plan.possible==1) & (plan['value']>0)].merge(pac_config,on='pack_id',how='left')
  plan2['qty_total'] = plan2['value'] * plan2.qty
  plan2.rename(columns={'variable':'store'},inplace=True)
  plan3 = plan2[['week_num','store','sku_id','qty_total']]

  # pivoting the plan to make it comparible with forecast
  plan4 = plan3.pivot_table(index=['week_num','sku_id'],columns='store'\
                            ,values='qty_total',aggfunc='sum',fill_value=0).reset_index()
  plan4.columns = ['plan_'+x if 'STR' in x else x for x in plan4.columns]
  plan4.rename(columns={'week_num':'week'},inplace=True)

  # comparison with forecast
  forecast_sel = forecast[['week','sku_id','STR0001','STR0002','STR0003','STR0004',\
                           'STR0005','STR0006','STR0007','STR0008','STR0009','STR0010']]
  merged_forecast_plan = forecast_sel.merge(plan4, how='left', on=['week','sku_id']).fillna(0)

  # checking for stockout
  for i in stores :
    # print(i)
    merged_forecast_plan['stockout_'+i] = merged_forecast_plan['plan_'+i] < merged_forecast_plan[i]

  stockout = merged_forecast_plan[['stockout_'+i for i in stores]]

  return merged_forecast_plan,stockout
```

Figure 6.1 : Generation Code for Comparison DataFrame and Stockout Table

```
stockout.sum().sum() / stockout.size
```

Figure 6.2 : Stockout Metric

Solution 1

STEPS For Part 2 :

1. Performed data wrangling to find each stores' highest selling stylecolor_id. (Figure 1.2)
2. Based on Figure 1.2, we can see that the styles that are the highest selling styles in each store is different
3. Finding the highest selling style is not actionable for the business team.
   Thus along with the highest selling styles, a plot to show the top selling styles, styles that are in the top 2% percentiles in terms of quantity sold in each store.(Figure 1.3)

   Insights from Figure 1.3 and the figures shared in the next page

Highest Style Sales in each Store

[36] Show code

	store_id	stylecolor_id	max_sold_by_store
3366	STR0001	4615284	3348.000
13439	STR0002	820615	4164.000
29696	STR0003	5117488	3346.000
42373	STR0004	4848421	3561.000
52632	STR0005	1158220	3650.000
70033	STR0006	7914848	3531.000
85725	STR0007	37771940	3374.000
98775	STR0008	43461812	3086.000
106085	STR0009	5075479	3568.000
123617	STR0010	29101804	3218.000

Figure 1.2 :
Table of Highest Selling Style at each Store