Data Generation - Sales

Abstract

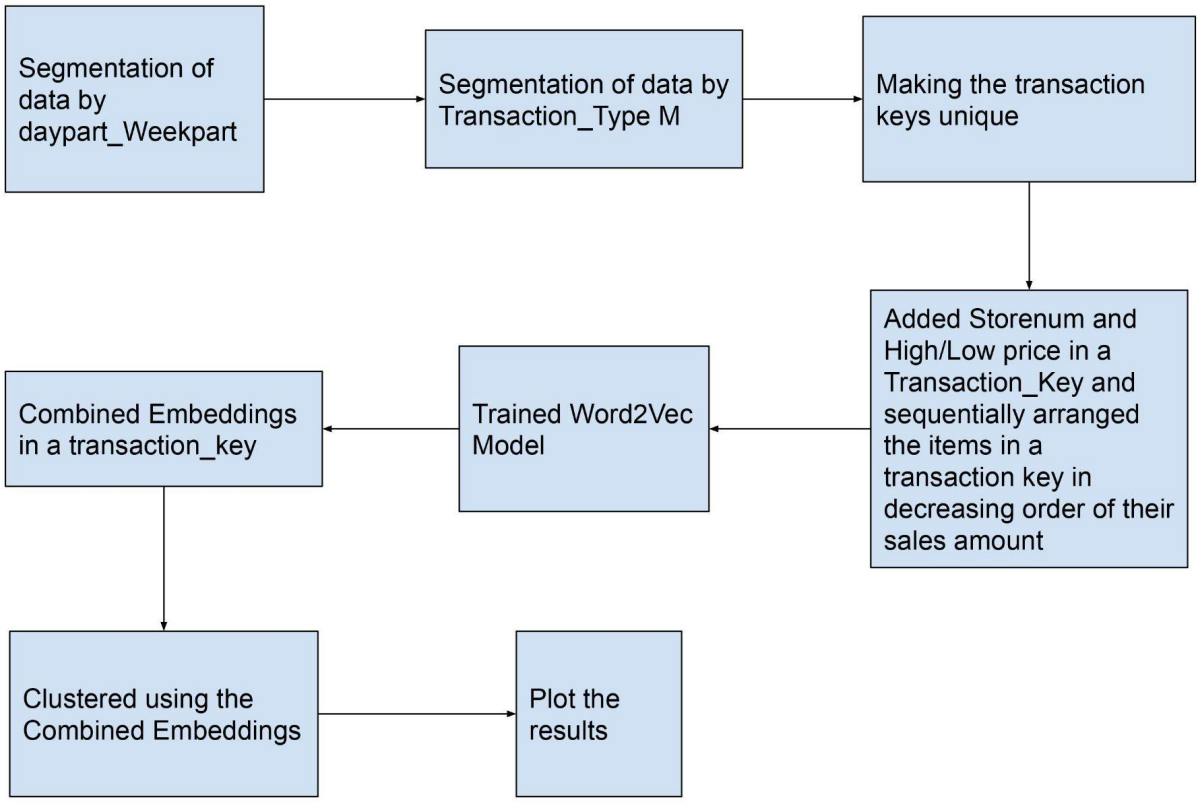
Food Companies often need to get alert when some changes occur in sales at a particular time of year. They often launch various campaigns and offers to attract customers. They need to estimate their expected sales and observe its dependent features during these campaigns. Also, some product testing, LTOs, promotional offers can be tested on some stores based on earlier transaction trends. So, generated data can be of great importance for companies' future behavior.

We worked on generation of sales data from a given set of features. Objective for that can be:

- To detect changes in sales from expected, to get alerted and take corresponding steps
- To detect effects on sales, if some changes occurs in features like storenum, sales channel

Clustering of Transactions

Procedure adopted



Flowchart : Clustering

Features used:

- Storenum
- Sales amount
- Date and time
- Transaction type
- Product description
- Sales channel

Method Description

- **Segment the dataset by daypart_Weekpart** - To begin with, we segmented the dataset based on transaction daypart alongwith part of the week i.e., Breakfast_Weekday, Dinner_Weekend, Lunch_Weekday etc.

  Dayparts were observed as:
  -> 00:00 to 12:00 - Breakfast
  -> 12:00 to 18:00 - Lunch
  -> 18:00 to 00:00 - Dinner

  Weekpart as:
  -> Monday to Friday - Weekday
  -> Saturday, Sunday - Weekend

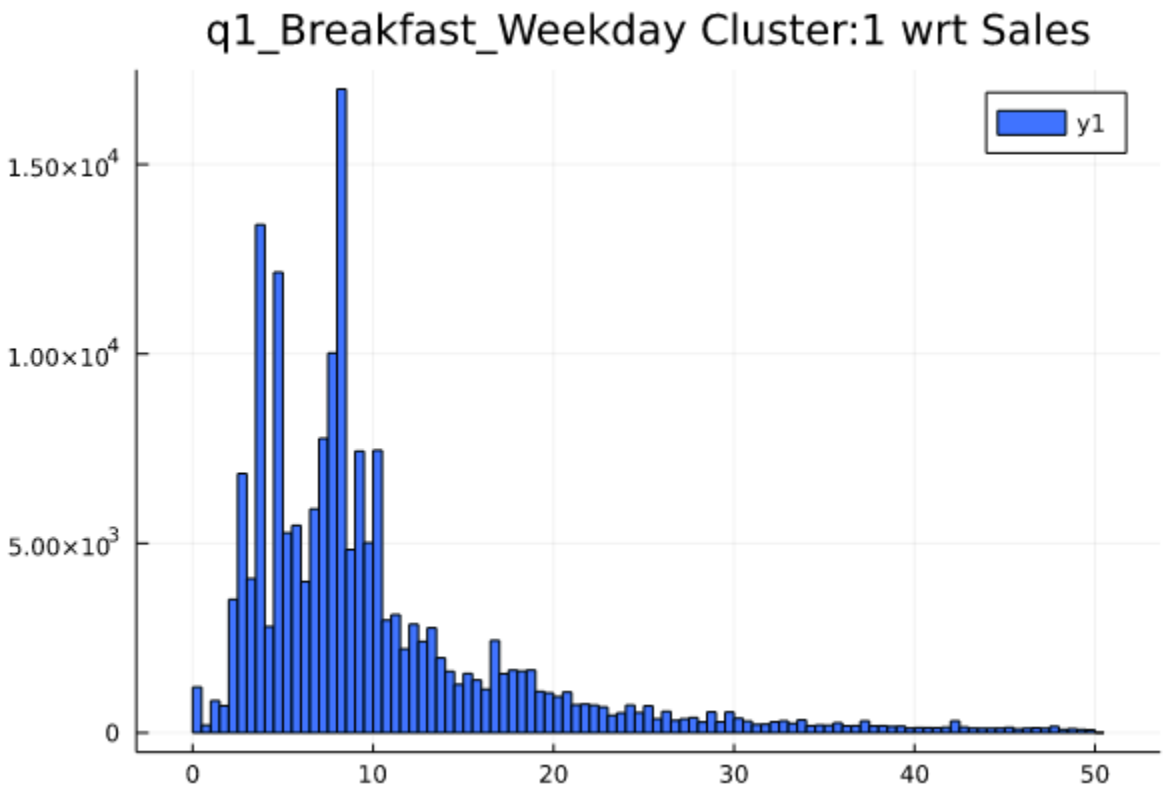
- **Segment only transaction type M**

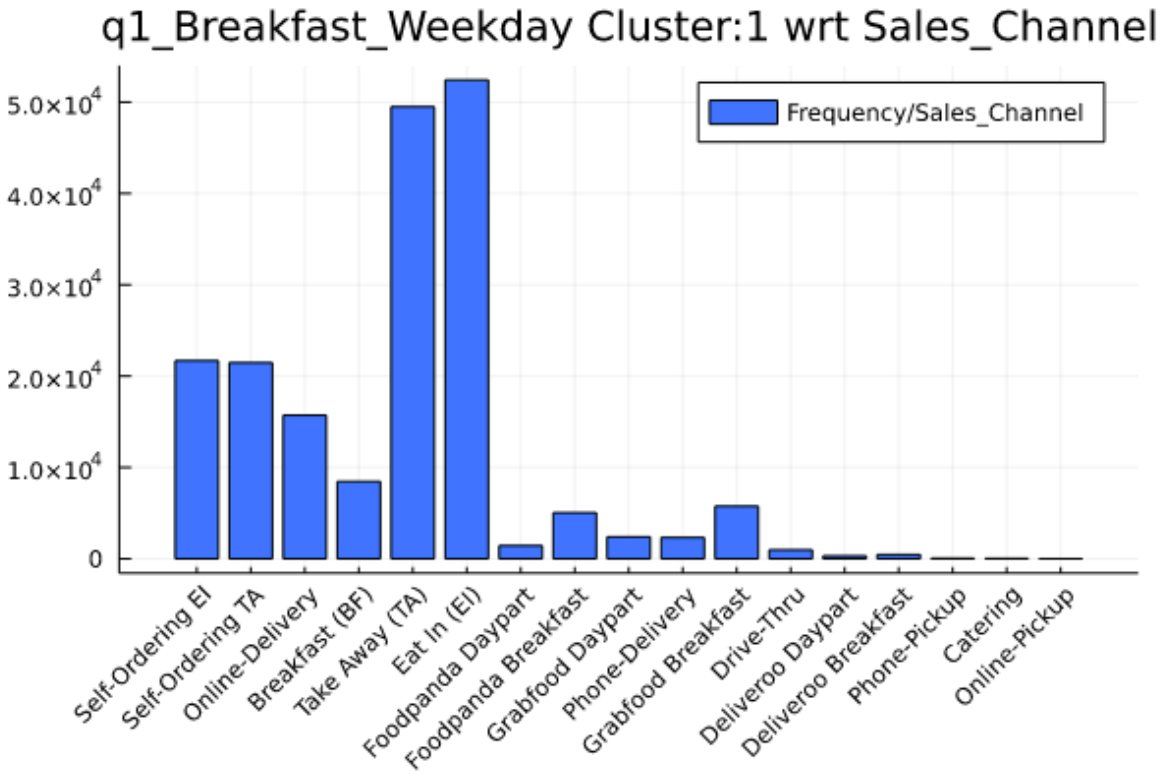
  M - Menu ; T - Tender ; D - Discount

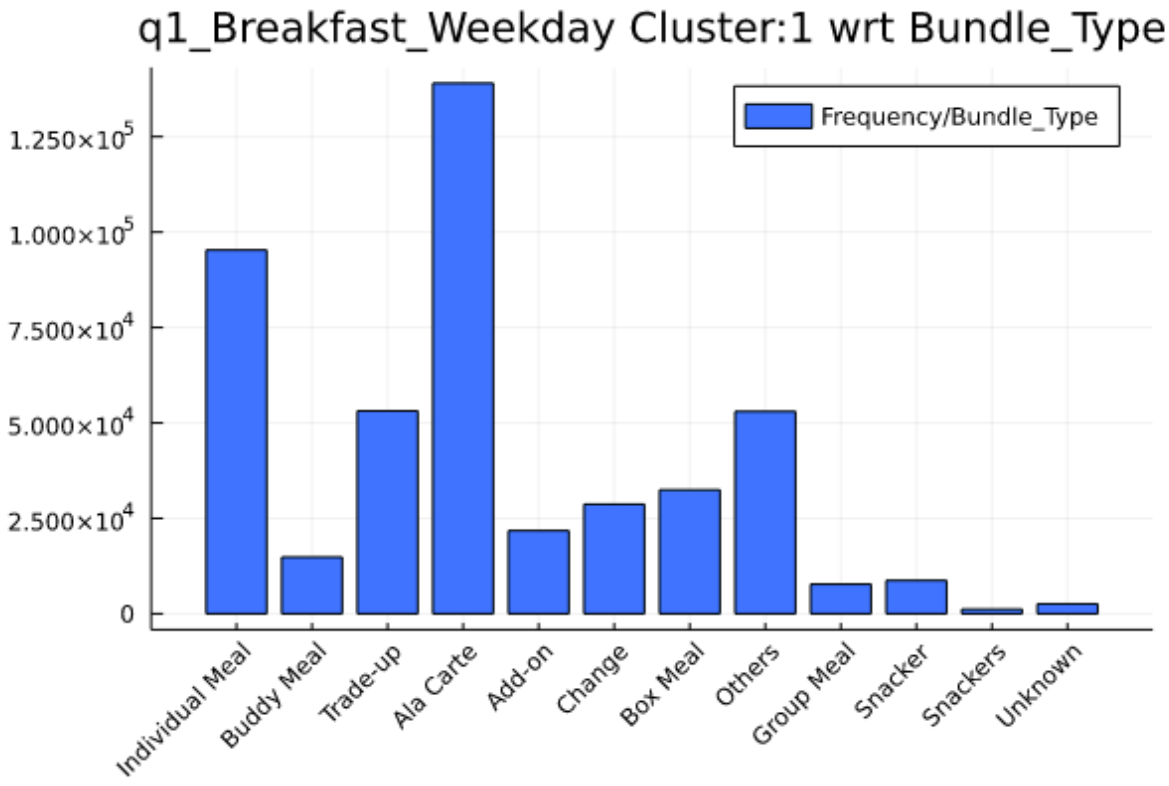
- **Make unique transaction keys** - Transaction keys were made unique by taking combination of storenum, transaction_key with format "storenum_transaction_key_date" and date of transaction as "storenum_transaction_key" were not unique and repeating after some days
- **Add storenum and High/Low price**(Highprice if sales amount < $15, else Lowprice) corresponding to unique transaction_key and sequentially arranged the items in transaction key in decreasing order of their sales amount
- **Trained Word2Vec model** with text file having storenum, High/Low price along with product descriptions space-separated ended with '\n'
- **Combined Embeddings** of product combination obtained after Wod2Vec training corresponding to each transaction_key
- **K-means clustering** was used to cluster the embeddings
- **Plot the results**

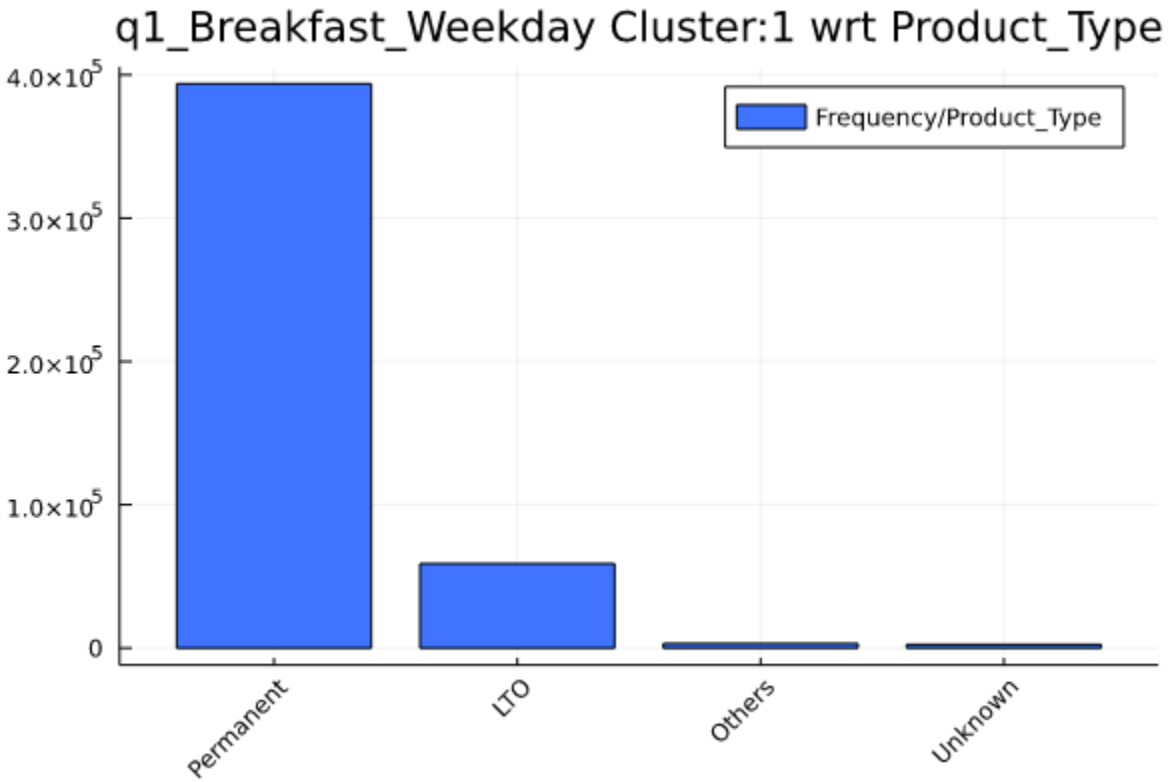
  Sales amount vs frequency
  Bundle type vs frequency
  Product type vs frequency
  sales channel vs frequency
  sales category vs frequency
  storenum vs frequency

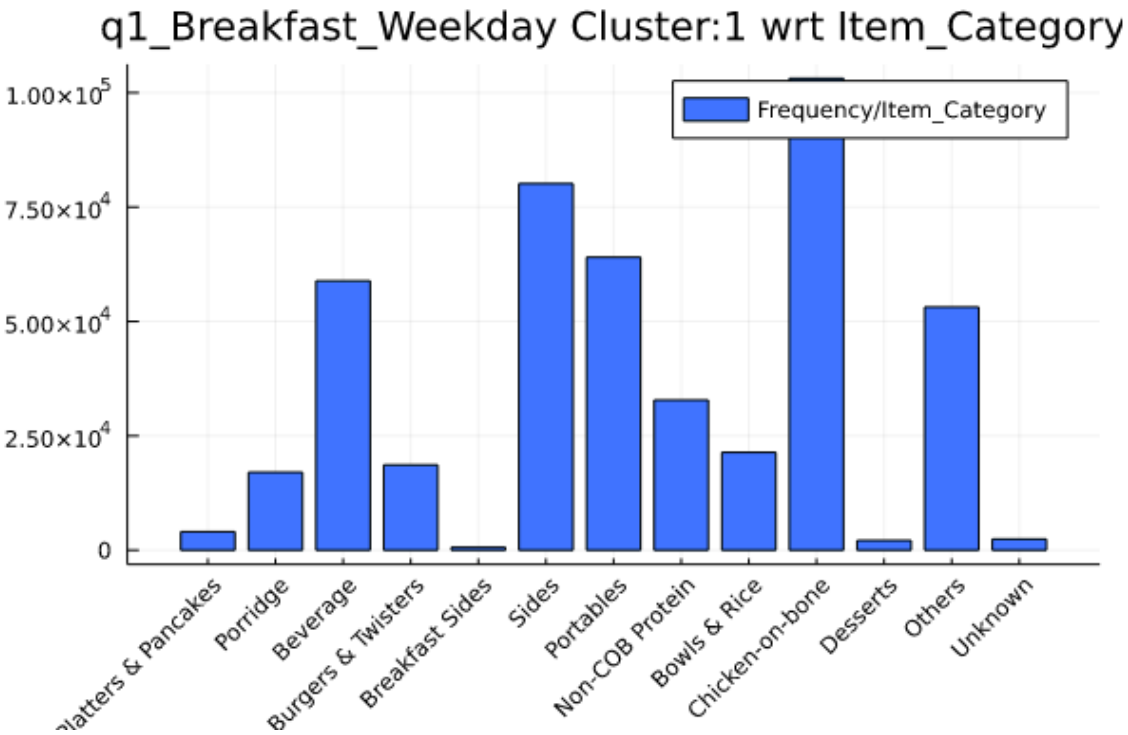
Results


q1_Breakfast_Weekday Cluster:1 wrt Sales
y1
1.50×10^4
1.00×10^4
5.00×10^3
0
0
10
20
30
40
50


q1_Breakfast_Weekday Cluster:1 wrt Sales_Channel
Frequency/Sales_Channel
5.0×10^4
4.0×10^4
3.0×10^4
2.0×10^4
1.0×10^4
0
Self-Ordering EI
Self-Ordering TA
Online-Delivery
Breakfast (BF)
Take Away (TA)
Eat In (EI)
Foodpanda Daypart
Foodpanda Breakfast
Grabfood Daypart
Phone-Delivery
Grabfood Breakfast
Drive-Thru
Deliveroo Daypart
Deliveroo Breakfast
Phone-Pickup
Catering
Online-Pickup

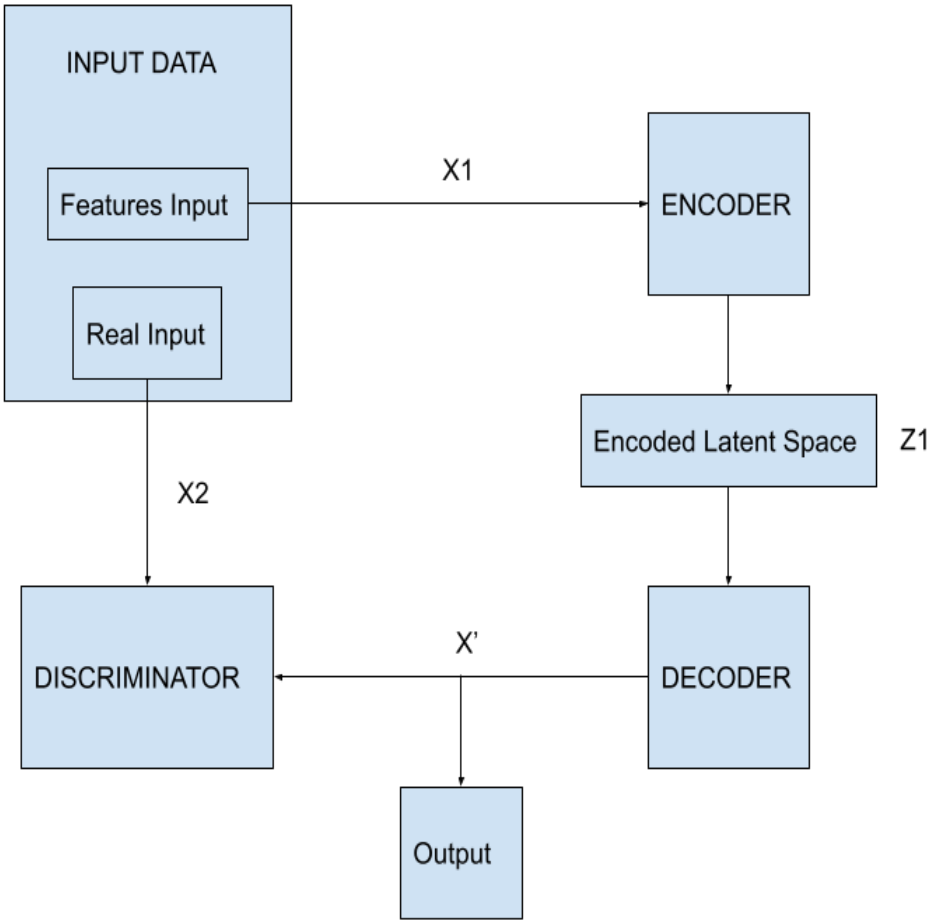

q1_Breakfast_Weekday Cluster:1 wrt Bundle_Type
Frequency/Bundle_Type
1.250×10^5
1.000×10^5
7.500×10^4
5.000×10^4
2.500×10^4
0
Individual Meal
Buddy Meal
Trade-up
Ala Carte
Add-on
Change
Box Meal
Others
Group Meal
Snacker
Snackers
Unknown


q1_Breakfast_Weekday Cluster:1 wrt Product_Type
Frequency/Product_Type
4.0×10^5
3.0×10^5
2.0×10^5
1.0×10^5
0
Permanent
LTO
Others
Unknown


q1_Breakfast_Weekday Cluster:1 wrt Item_Category
Frequency/Item_Category
1.00×10^5
7.50×10^4
5.00×10^4
2.50×10^4
0
Platters & Pancakes
Porridge
Beverage
Burgers & Twisters
Breakfast Sides
Sides
Portables
Non-COB Protein
Bowls & Rice
Chicken-on-bone
Desserts
Others
Unknown

GENERATIVE PART

Sales_Data_Generation Overview



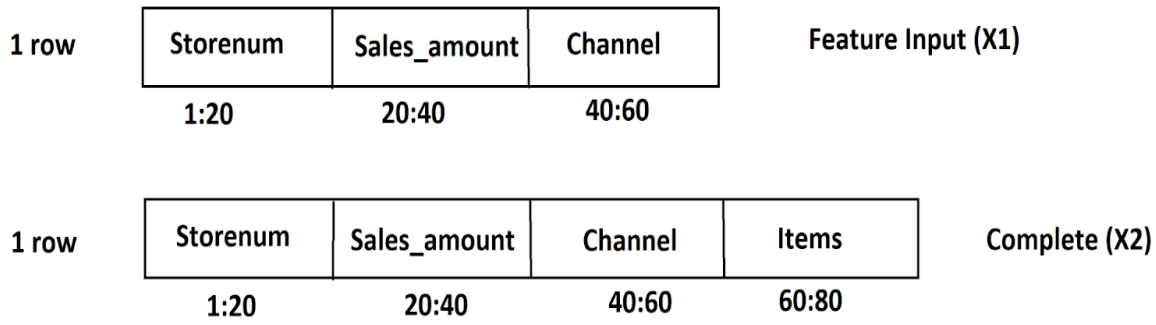
X1 : Features Input embeddings
X2 : Complete embeddings
X' : Generated Data (Combined embedding of a Transaction)
Z1 : Latent Space of X

Pipeline motivation received from https://arxiv.org/pdf/1805.06725.pdf

Input Format

Input data is divided into 2 parts :

- X1 - > Feature embeddings contain the concatenated embeddings of the features (Store number + Sales amount + Sales Channel) of a transaction .
- X2 - > Complete embeddings contain the feature embeddings concatenated with the combined embeddings of the items in a transaction



- 1 Input element contains data_y number of rows.

Pipeline

Framework : Flux

- https://fluxml.ai/Flux.jl/stable/models/overview/

Encoder :

- Input : Features input (dims = 60*data_y)
- First layer is LSTM which is followed by Dense layers
- No Activation functions used (Tried Relu , Leakyrelu and Tanh Activation in the last layer but none of them improved the performance)
- Output : Matrix of dims = 40*10
- Range of Matrix elements (-1 to 1)

Noise Vector :

- Converted to Range of (-1 to 1)
- Dims = 40*10
- Concatenated with Encoder output , Resultant Matrix dims = 40*20

Decoder :

- Input : Matrix of dims = 40*20

- Combination of Dense Layers only
- No Activation function used (Tried Relu , Leakyrelu and Tanh Activation in the last layer but none of them improved the performance)
- Output : Matrix of dims = 60*data_y

Discriminator :

- Input : Matrix of dims = 80*data_y and label (real -> 1 / fake -> 0) , One batch of Discriminator input contains half batch taken from Real data (X2) and half batch taken from Decoder Output
- Combination of Dense Layers
- Preferred Activation : Sigmoid in the last layer (but got better results after removing it)
- Output : Single Floating point number

Optimizers :

- Tried with ADAM() , AdaBoost() , Adamax() and RMSProp()
- RMSProp() with learning rate as 0.001 and momentum as 0.95 for all the updates gave the best results.
- https://fluxml.ai/Flux.jl/stable/training/optimisers/

Losses :

- Discriminator Loss : Loss(Discriminator(X2),1) + Loss(Discriminator(X'),0)
  Best result from LogitBinaryCrossentropy
- Decoder Loss : Loss(Discriminator(X') , 1)
  Best result from LogitBinaryCrossentropy
- Contextual Loss : Loss(X1 , X')
  Best result from MSE
- In all the above Losses , we tried various Loss Functions from https://fluxml.ai/Flux.jl/stable/models/losses/

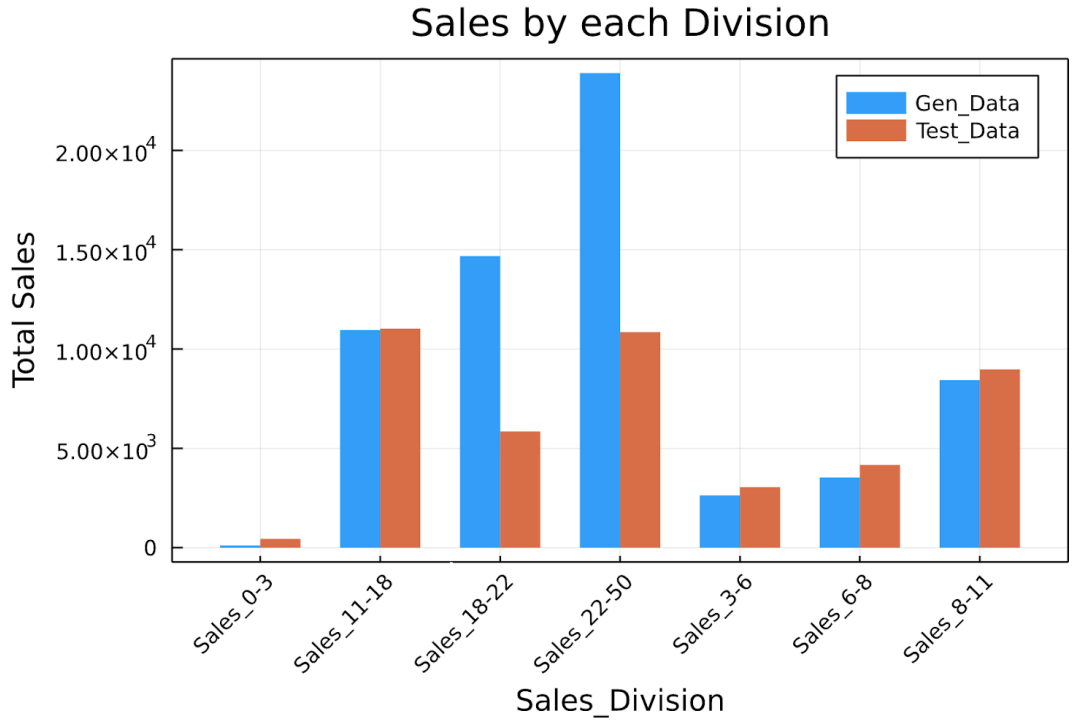
Output Generation :

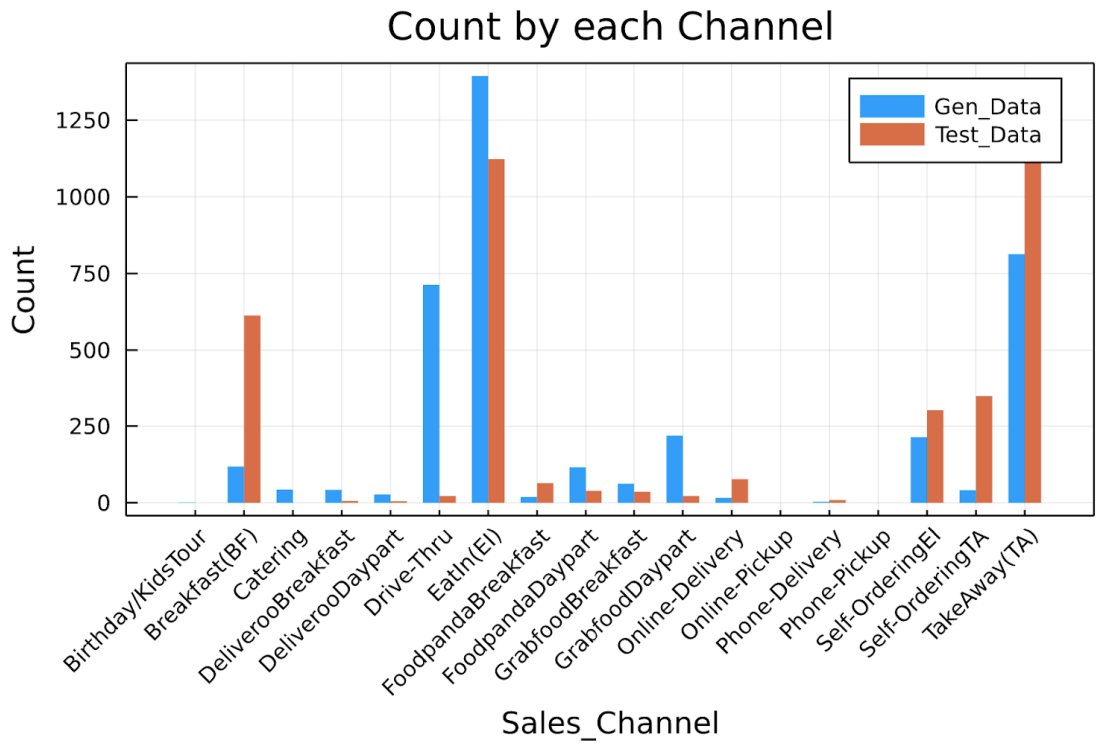
- Each row in Generated Matrix from Decoder represents a transaction
- Cosine Distance of each row is measured from the transaction embeddings of all transactions belonging to their respective segments.
- Transaction details of most similar transaction are fetched from the data and culminated in the form of Generated Sales data on which several analysis can be performed to check the performance of the model

Results

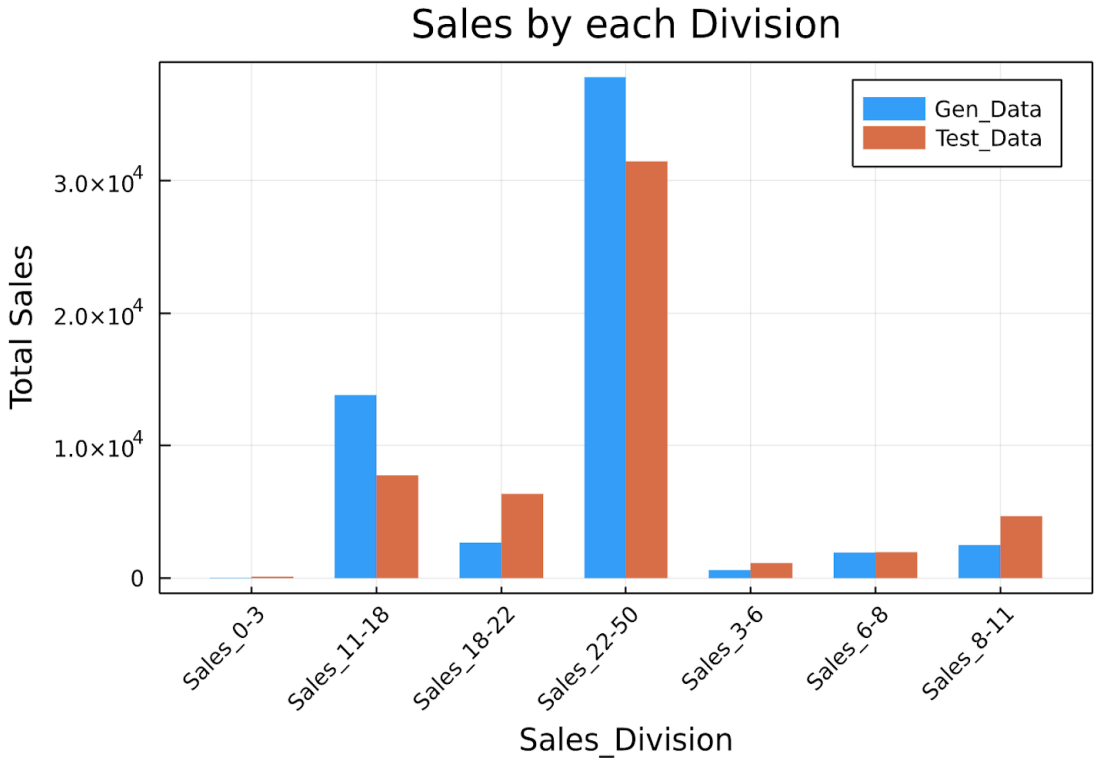
Model was trained on 4 quarters in a year separately and hence we have quarter wise results :

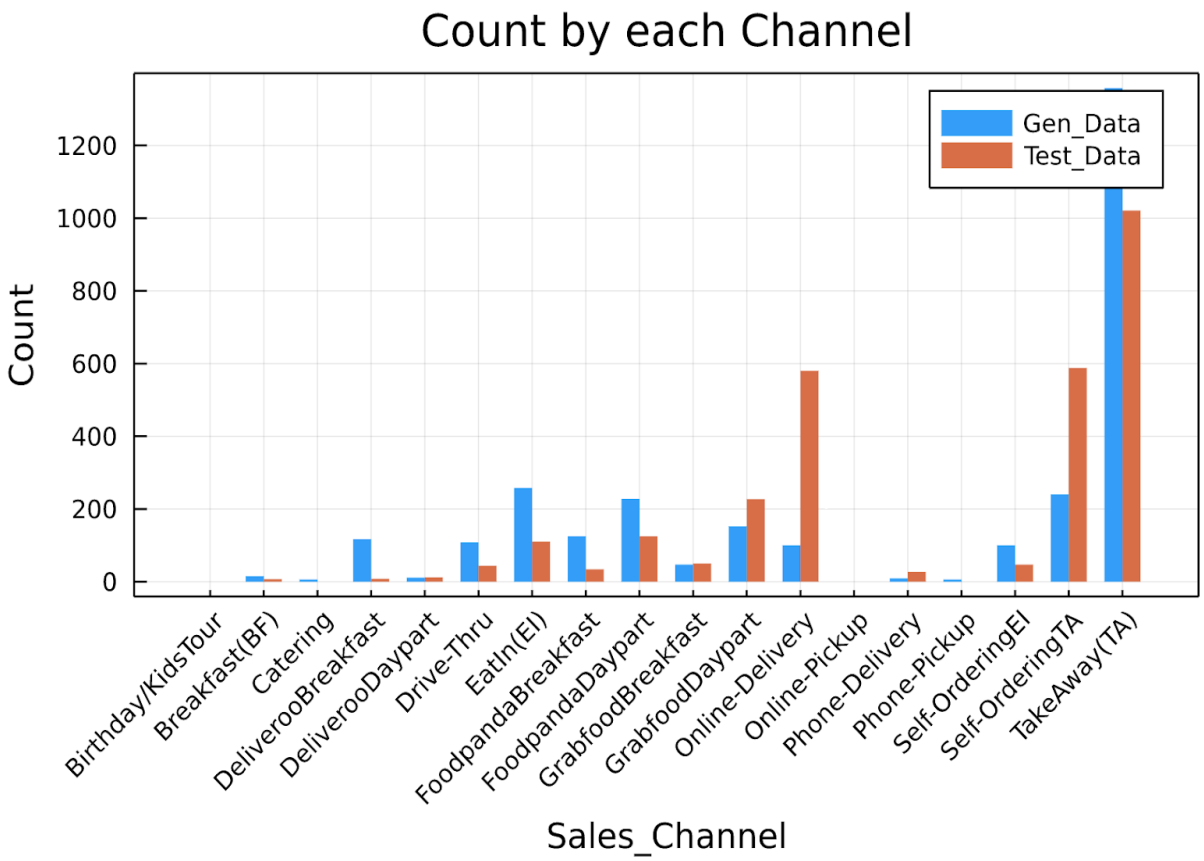
Quarter 1 : Jan - Mar



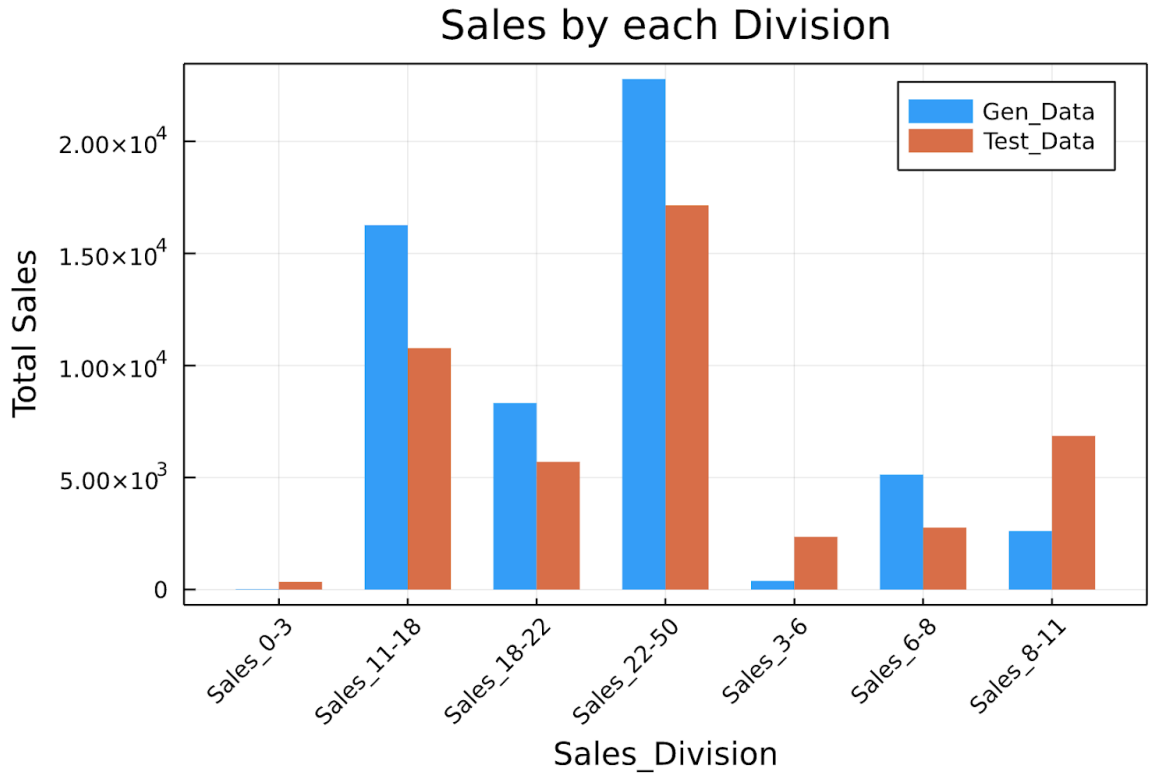


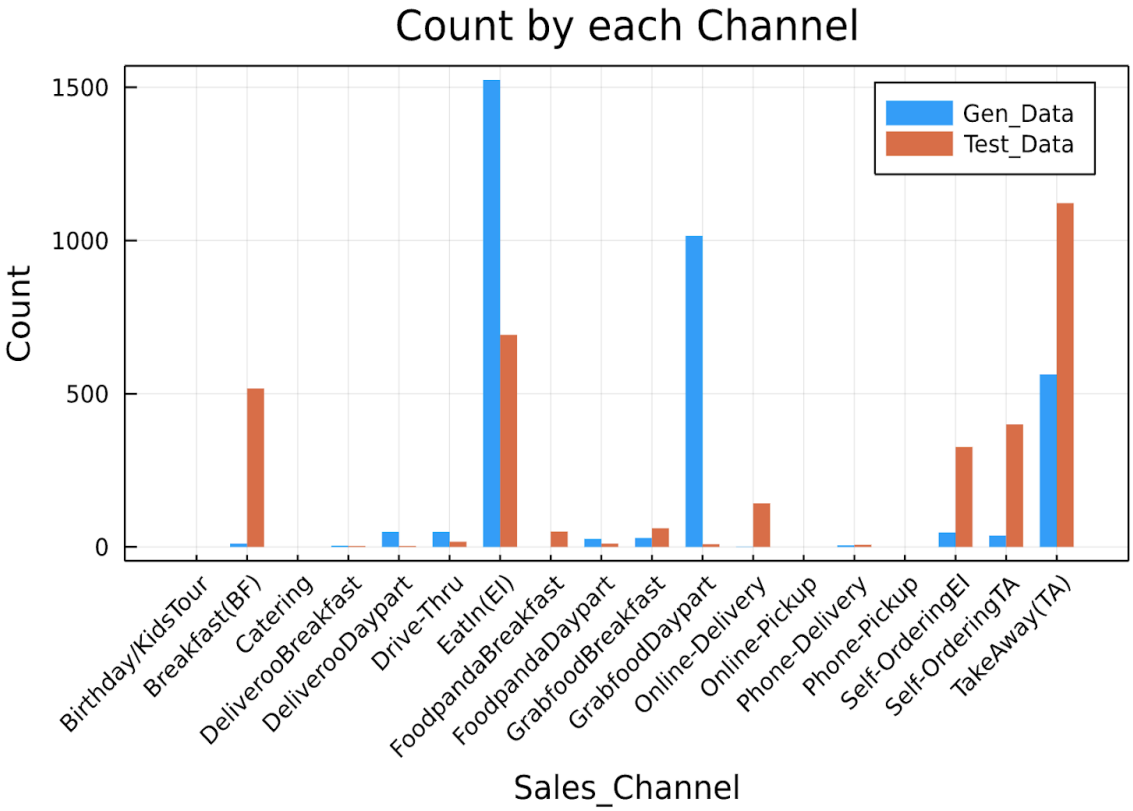
Quarter 2: Apr - Jun



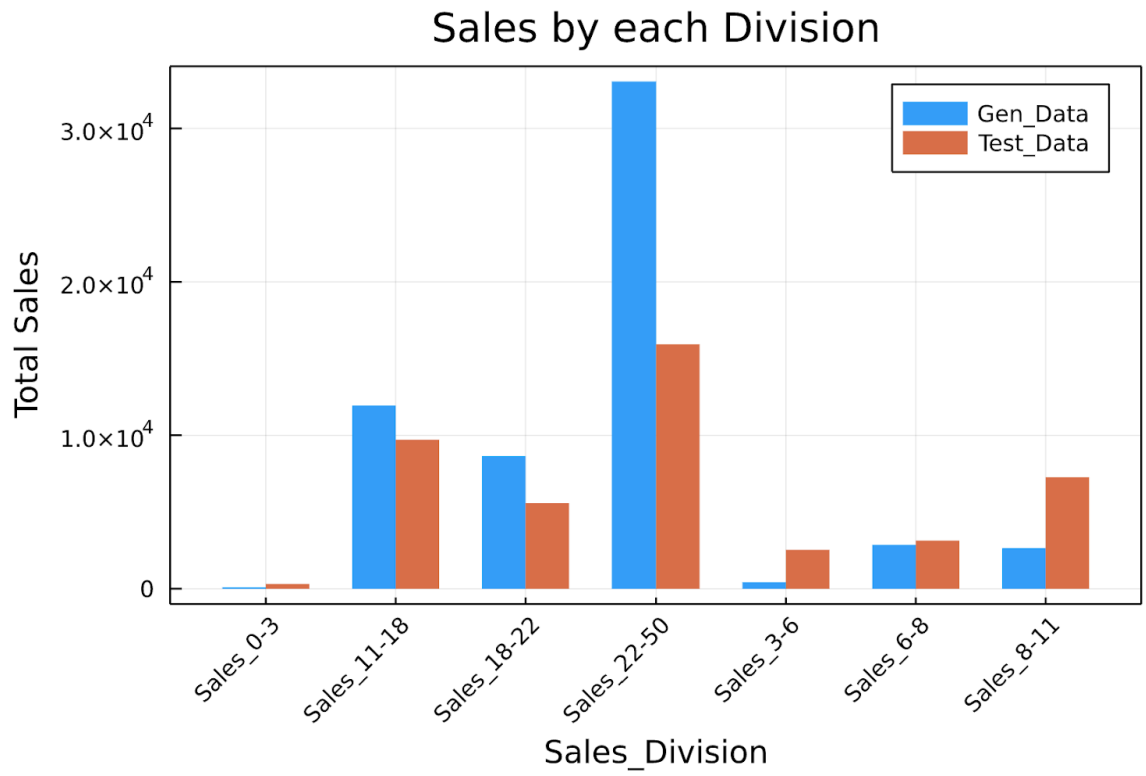


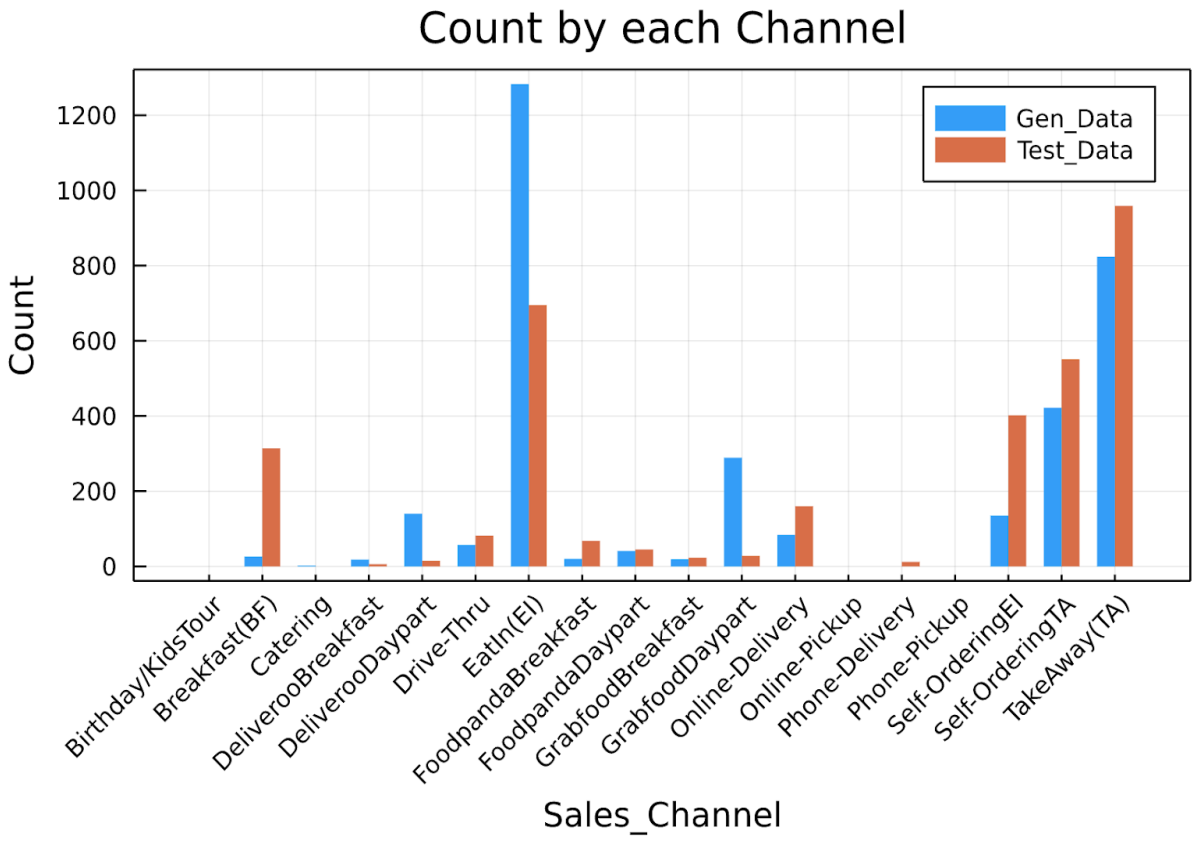
Quarter 3 : Jul - Sep





Quarter 4 : Oct - Dec





Observation :

- Performance in terms of Sales Division is better than compared to the performance in terms of Sales Channel
- Quarter 2 Results comes out to be the best of all the Quarters
- Sales Division (22-50) is the main issue
- No particular deviating Sales Channel could be identified . All the Channels show a significant difference in 1 quarter or the other.

Suggestions to improve the results :

- *Changing the embeddings of the Training data :*
  Embeddings of any 2 channels are not very far away (similar in terms of cosine distance) due to which the model is facing difficulties in learning the context accurately . Using Embeddings which are quite different from each other could be useful .

- *Changing the dimensions of the embeddings used :*
  Currently embedding for each feature is of dimension = 20 .Using some other dimension in this case (preferably large like 100) could be useful .

- *Organizing the dataset properly :*
  Some Sales channels tend to get lost if their count in the training data is very less and they are mostly surrounded by those channels which have much higher frequency . In this case , the model is unable to learn those specific channels . So , organizing the data in a more ordered way where each feature of a transaction is significant could help the model learn all the features in less amount of time .

- *Changing the format of Input :*
  We have used the embeddings to represent a particular transaction along with its features . But some other form of representation like Probabilistic distributions could also be used which could make it easier for the model to learn all the features effectively.

Challenges

- Smaller matrix dimension should be avoided as it would result in duplication of output elements by the Decoder

- Addition of noise vector to Encoder output was done to remove duplication of transactions within the same Output Element

- Range of Encoder Output does not play a significant role but its dimension is very crucial in the performance of the model and is also useful in removing duplication in the output.

- Training time is inversely proportional to the number of transactions in one input element (data_y) i.e. increasing data_y will decrease the overall training time .Data_y we have used : 10 , 20 , 40 , 80 , 240 , 480 .Larger the training data , larger should be the value of data_y .But , larger the data_y , more difficult it will become for the Decoder and Discriminator to learn characteristics of the training set .Increasing data_y should be accompanied with lower learning rate and higher number of epochs.

- We have used Parallel Computing only while training the model by enabling Multi-Threading . In this method , all the cores are used for the execution of the process but these processes are not sequential in nature . So, dynamically changing parameters based on the number of epochs should be avoided while using the concept of Parallel Computing. We created an environment where the number of threads are made equal to the number of cores available (16 in our case) [https://discourse.julialang.org/t/enable-multiple-cores-for-jupyter-lab/18658/7]. Then , by using threads macro [https://docs.julialang.org/en/v1/manual/multi-threading/] before the training loop , we were able to use all the 16 cpu simultaneously resulting in greater than 1500% cpu usage .

Limitations

- Training of Word2Vec Model using Word2Vec.jl package needs to be only executed on Linux based OS. However , loading a Pre-Trained Word2Vec model and further using it on any OS is possible.
- Some transaction keys have negative sales number and some having greater than $50, these keys were ignored to discard anomalies
- Only transaction type M was used because in some transactions only type M was present with no corresponding transaction type T
- Dynamic changes to the training loop could not be performed due to the use of Multithreading Concept as the loops are not executed sequentially
- The large value of data_y should be avoided as it will result in duplication of results

References:

- https://github.com/JuliaText/Word2Vec.jl
- https://github.com/JuliaStats/Clustering.jl
- https://www.mckinsey.com/business-functions/marketing-and-sales/our-insights/unlocking-the-power-of-data-in-sales#
- https://www.linkedin.com/pulse/data-science-sales-marketing-siddharth-vijayvergiya