National College of Ireland

Configuration Manual

MSc Research Project
Programme Name

Boryana Mihaylova
Student ID: x23295546

School of Computing
National College of Ireland

Supervisor: Mohammed Hasanuzzaman

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Boryana Mihaylova
Student ID:	X23295546
Programme:	MSc Data Analytics **Year:** 2
Module:	MSc Research Project
Lecturer:	Mohammed Hasanuzzaman
Submission Due Date:	11 December 2025
Project Title:	Exploring the impact of customer demographics and coupon redemption on the accuracy of XGBoost demand forecasting model
Word Count:	..... **Page Count:** .....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Boryana Mihaylova

Date: 11/12/2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

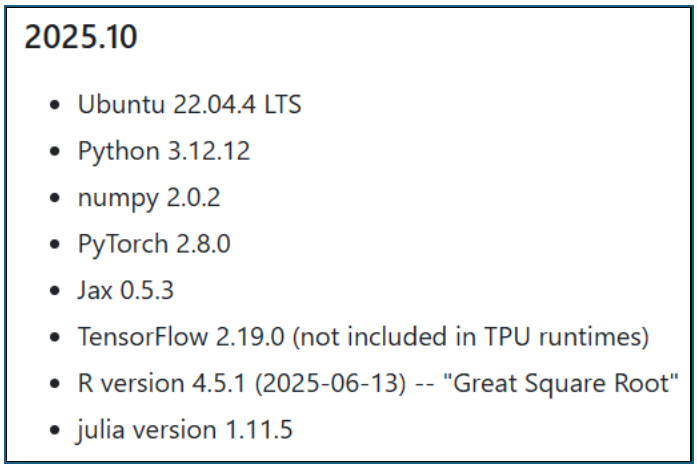
Configuration Manual

Forename Surname
Student ID:

1 System Configuration

Your first section. Change the header and label to something appropriate. The project was implemented in Google Colab in the Python programming language.

Tier: Colab Pro
Hosted runtime: High-RAM / 51 GB
Runtime version



2 Library Packages

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import datetime as dt

# from bayes_opt import BayesianOptimization
from sklearn import model_selection
from sklearn.model_selection import train_test_split
from sklearn import linear_model
from sklearn import metrics
from sklearn import neighbors
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

import xgboost as xgb
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

I have also imported the shap library.

3 Data Understanding

3.1 Exploratory data analysis of tables before merging

3.1.1 Transactions

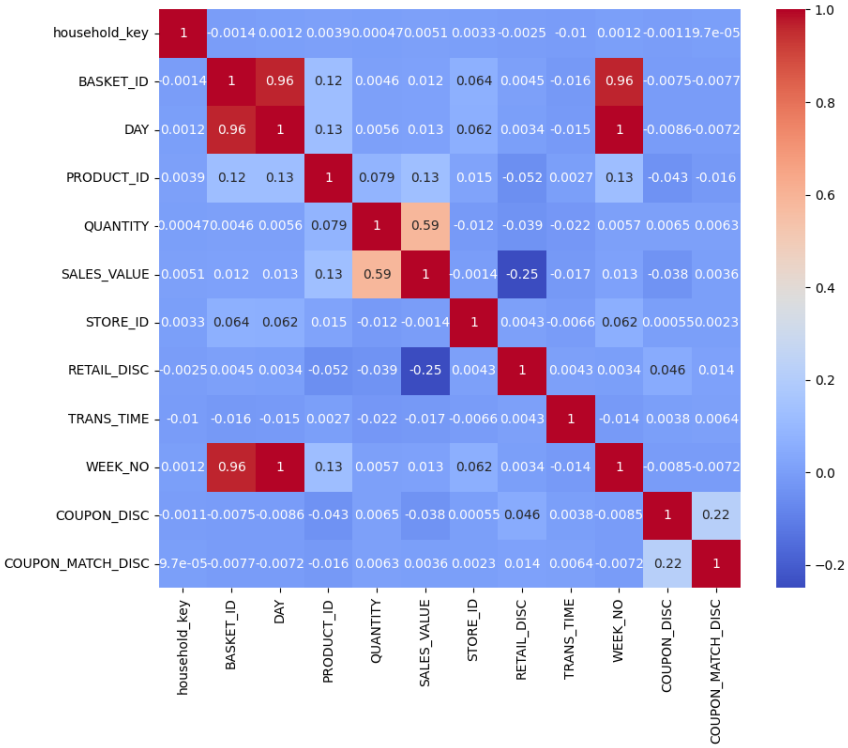
```
# check the number of rows and columns
transactions.shape
```

```
(2595732, 12)
```

```
# columns info
transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2595732 entries, 0 to 2595731
Data columns (total 12 columns):
 #   Column             Dtype  
---  ------             -----  
 0   household_key      int64  
 1   BASKET_ID          int64  
 2   DAY                int64  
 3   PRODUCT_ID         int64  
 4   QUANTITY           int64  
 5   SALES_VALUE        float64
 6   STORE_ID           int64  
 7   RETAIL_DISC        float64
 8   TRANS_TIME         int64  
 9   WEEK_NO            int64  
 10  COUPON_DISC        float64
 11  COUPON_MATCH_DISC  float64
dtypes: float64(4), int64(8)
memory usage: 237.6 MB
```

Correlation matrix



3.1.2 Products

```
# read file
products = pd.read_csv('product.csv')
```

```
# check the number of rows and columns
products.shape
```

```
(92353, 7)
```

```
# column info
products.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 92353 entries, 0 to 92352
Data columns (total 7 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   PRODUCT_ID            92353 non-null  int64 
 1   MANUFACTURER          92353 non-null  int64 
 2   DEPARTMENT            92353 non-null  object
 3   BRAND                 92353 non-null  object
 4   COMMODITY_DESC        92353 non-null  object
 5   SUB_COMMODITY_DESC    92353 non-null  object
 6   CURR_SIZE_OF_PRODUCT  92353 non-null  object
dtypes: int64(2), object(5)
memory usage: 4.9+ MB
```

```
products['PRODUCT_ID'].nunique()
```

```
92353
```

```
# No. of manufacturers
products['MANUFACTURER'].nunique()
```

```
6476
```

```
# No. of departments
products['DEPARTMENT'].nunique()
```

```
44
```

```
# No. of brands
products['BRAND'].nunique()
```

```
2
```

```
# No. of unique commodity description
products['COMMODITY_DESC'].nunique()
```

```
308
```

```
# No. of unique sub-commodity descriptions
products['SUB_COMMODITY_DESC'].nunique()
```

```
2383
```

```
# No. of unique current sizes of product
products['CURR_SIZE_OF_PRODUCT'].nunique()
```

```
4345
```

3.1.3 Coupons

Read the file | Shape | Column Info

```
# read file
coupon = pd.read_csv('coupon.csv')
```

```
# shape
coupon.shape
```

```
(124548, 3)
```

```
# columns
coupon.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 124548 entries, 0 to 124547
Data columns (total 3 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   COUPON_UPC  124548 non-null  int64
 1   PRODUCT_ID  124548 non-null  int64
 2   CAMPAIGN    124548 non-null  int64
dtypes: int64(3)
memory usage: 2.9 MB
```

```
# first five rows
coupon.head()
```

	COUPON_UPC	PRODUCT_ID	CAMPAIGN
0	10000089061	27160	4
1	10000089064	27754	9
2	10000089073	28897	12
3	51800009050	28919	28
4	52100000076	28929	25

3.1.4 Coupon Redemption

Read the file | Shape | Column Info

```
#read the file
coupon_redempt = pd.read_csv('coupon_redempt.csv')
```

```
# check the number of rows and columns (shape)
coupon_redempt.shape
```

```
(2318, 11)
```

```
# columns info
coupon_redempt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2318 entries, 0 to 2317
Data columns (total 11 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   household_key  2318 non-null   int64  
 1   DAY            2318 non-null   int64  
 2   COUPON_UPC     2318 non-null   int64  
 3   CAMPAIGN       2318 non-null   int64  
 4   Unnamed: 4     0 non-null      float64
 5   Unnamed: 5     0 non-null      float64
 6   Unnamed: 6     0 non-null      float64
 7   Unnamed: 7     0 non-null      float64
 8   Unnamed: 8     0 non-null      float64
 9   Unnamed: 9     0 non-null      float64
 10  Unnamed: 10    0 non-null      float64
dtypes: float64(7), int64(4)
memory usage: 199.3 KB
```

Stats

```
# No. of households who redeemed a coupon
coupon_redempt['household_key'].nunique()
```

```
434
```

```
# No. of redeemed coupons
coupon_redempt['COUPON_UPC'].nunique()
```

```
556
```

```
# No of unique campaigns
coupon_redempt['CAMPAIGN'].nunique()
```

```
30
```

```
# No. of days
coupon_redempt['DAY'].nunique()
```

```
328
```

**Important note: ** In the transaction data table, coupons were redeemed on 705 days out of the 711 days, while the coupon_redempt only has data for 328 days. In addition, in transaction data 1858 unique id availed of coupon discount while here we see only 434.

```
# minimum day
coupon_redempt['DAY'].min()
```

```
225
```

```
# maximum day
coupon_redempt['DAY'].max()
```

```
704
```

```
# check for duplicates in coupon_redempt
coupon_redempt.duplicated().sum()
```

```
np.int64(0)
```

3.1.5 Demographics

Read the file | Shape | Columns | Head

```
# read the file
demographics = pd.read_csv('hh_demographic.csv')
```

```
# shape
demographics.shape
```

```
(801, 8)
```

```
# No. of rows and columns
demographics.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 801 entries, 0 to 800
Data columns (total 8 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   AGE_DESC             801 non-null    object
 1   MARITAL_STATUS_CODE  801 non-null    object
 2   INCOME_DESC          801 non-null    object
 3   HOMEOWNER_DESC       801 non-null    object
 4   HH_COMP_DESC         801 non-null    object
 5   HOUSEHOLD_SIZE_DESC  801 non-null    object
 6   KID_CATEGORY_DESC    801 non-null    object
 7   household_key        801 non-null    int64 
dtypes: int64(1), object(7)
memory usage: 50.2+ KB
```

Stats

```
# check if household key is unique
demographics['household_key'].nunique()
```

```
801
```

```
# No. of unique age descriptions
demographics['AGE_DESC'].nunique()
```

```
6
```

```
# print the age descriptions
ages = demographics['AGE_DESC'].unique()
print(ages)
```

```
['65+' '45-54' '25-34' '35-44' '19-24' '55-64']
```

```
# count of household_key by age_desc
demographics['AGE_DESC'].value_counts()
```

AGE_DESC	count
45-54	288
35-44	194
25-34	142
65+	72
55-64	59
19-24	46

dtype: int64

```
# count of household keys by income_desc
demographics['INCOME_DESC'].value_counts()
```

INCOME_DESC	count
50-74K	192
35-49K	172
75-99K	96
25-34K	77
15-24K	74
Under 15K	61
125-149K	38
100-124K	34
150-174K	30
250K+	11
175-199K	11
200-249K	5

dtype: int64

```
# count of household key by homeowner_desc
demographics['HOMEOWNER_DESC'].value_counts()
```

HOMEOWNER_DESC	count
Homeowner	504
Unknown	233
Renter	42
Probable Renter	11
Probable Owner	11

dtype: int64

```
# count of households by hh_comp_desc
demographics['HH_COMP_DESC'].value_counts()
```

HH_COMP_DESC	count
2 Adults No Kids	255
2 Adults Kids	187
Single Female	144
Single Male	95
Unknown	73
1 Adult Kids	47

dtype: int64

```
# count of households by hh size desc
demographics['HOUSEHOLD_SIZE_DESC'].value_counts()
```

HOUSEHOLD_SIZE_DESC	count
2	318
1	255
3	109
5+	66
4	53

KID_CATEGORY_DESC	count
None/Unknown	558
1	114
3+	69
2	60

3.1.6 Campaigns

Read the file | Shape | Info | Head

```
# read the file
campaign = pd.read_csv('campaign_table.csv')
```

```
# shape
campaign.shape
```

```
(7208, 3)
```

```
# columns
campaign.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7208 entries, 0 to 7207
Data columns (total 3 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   DESCRIPTION    7208 non-null   object
 1   household_key  7208 non-null   int64 
 2   CAMPAIGN       7208 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 169.1+ KB
```

```
# print first five rows of the table
campaign.head()
```

	DESCRIPTION	household_key	CAMPAIGN
0	TypeA	17	26
1	TypeA	27	26
2	TypeA	212	26
3	TypeA	208	26
4	TypeA	192	26

```
# No. of unique campaigns
campaign['CAMPAIGN'].nunique()
```

```
30
```

```
# Unique campaign types
campaign['DESCRIPTION'].nunique()
```

```
3
```

```
# campaign types
campaign_type = campaign['DESCRIPTION'].unique()
print(campaign_type)
```

```
['TypeA' 'TypeB' 'TypeC']
```

```
# check for duplicates in campaign
campaign.duplicated().sum()
```

```
np.int64(0)
```

3.1.7 Campaign description table

Read the file, shape, info, head

```
# read the file
campaign_desc = pd.read_csv('campaign_desc.csv')
```

```
# campaign description shape
campaign_desc.shape
```

```
(30, 4)
```

```
# columns
campaign_desc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   DESCRIPTION  30 non-null     object
 1   CAMPAIGN     30 non-null     int64 
 2   START_DAY    30 non-null     int64 
 3   END_DAY      30 non-null     int64 
dtypes: int64(3), object(1)
memory usage: 1.1+ KB
```

4 Feature Engineering

4.1 RFM

I have used the RFM_Score metric directly without creating segments such as: loyal customer, champion or lost.

Recency, Frequency, Monetary Value

[Customer Segmentation using RFM Analysis]

```
merged = pd.read_csv('transactions1.csv')
merged.head()
```

	household_key	BASKET_ID	DAY	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID	RETAIL_DISC	TRANS_TIME	WEEK_NO	...	BRAND	DEPAR
0	1364	26984896261	1	842930	1	2.19	31742	0.00	1520	1	...	Private	GRO
1	1364	26984896261	1	897044	1	2.99	31742	-0.40	1520	1	...	National	GRO
2	1364	26984896261	1	920955	1	3.09	31742	0.00	1520	1	...	National	
3	1364	26984896261	1	937406	1	2.50	31742	-0.99	1520	1	...	National	P
4	1364	26984896261	1	981760	1	0.60	31742	-0.79	1520	1	...	Private	GRO

5 rows × 25 columns

```
# rfm analysis, group by household, columns: max day, unique count of baskets, total sales
rfm_table = merged.groupby('household_key').agg(
    MostRecentDay=('DAY', 'max'),
    Frequency=('BASKET_ID', 'nunique'),
    Monetary=('SALES_VALUE', 'sum')
)
rfm_table.head()
```

household_key	MostRecentDay	Frequency	Monetary
1	706	86	4330.16
7	709	59	3400.05
8	706	113	5534.97
13	709	275	13190.92
16	690	98	1512.02

```
# calculate the recency for each household: MostRecentDay.max - MostRecentDay
rfm_table['Recency'] = rfm_table['MostRecentDay'].max() - rfm_table['MostRecentDay']
rfm_table.head(20)
```

household_key	MostRecentDay	Frequency	Monetary	Recency
1	706	86	4330.16	5
7	709	59	3400.05	2
8	706	113	5534.97	5
13	709	275	13190.92	2
16	690	98	1512.02	21
17	689	124	5139.63	22
18	711	163	7118.91	0

```
# add monetary rank column from 1-5 using monetary column. Higher amount means high rank
# https://pandas.pydata.org/docs/reference/api/pandas.qcut.html
rfm_table['MonetaryRank'] = pd.qcut(rfm_table['Monetary'], 5, labels=range(1, 6))
rfm_table.head()
```

household_key	MostRecentDay	Frequency	Monetary	Recency	MonetaryRank
1	706	86	4330.16	5	3
7	709	59	3400.05	2	2
8	706	113	5534.97	5	4
13	709	275	13190.92	2	5
16	690	98	1512.02	21	1

```
# add frequency rank from 1-5 based on the Frequency column. higher value means high rank
# handle duplicates with rank
# https://medium.com/@heyamit10/resolving-issues-with-duplicates-in-pandas-qcut-c589fdb56703
rfm_table['FrequencyRank'] = pd.qcut(rfm_table['Frequency'].rank(method='first'), 5, labels=range(1, 6))
rfm_table.head()
```

household_key	MostRecentDay	Frequency	Monetary	Recency	MonetaryRank	FrequencyRank
1	706	86	4330.16	5	3	1
7	709	59	3400.05	2	2	1
8	706	113	5534.97	5	4	2
13	709	275	13190.92	2	5	5
16	690	98	1512.02	21	1	2

```
# add recency rank, lower recency value --> higher rank
rfm_table['RecencyRank'] = pd.qcut(rfm_table['Recency'].rank(method='first'), 5, labels=range(5, 0, -1))
rfm_table.head()
```

household_key	MostRecentDay	Frequency	Monetary	Recency	MonetaryRank	FrequencyRank	RecencyRank
1	706	86	4330.16	5	3	1	2
7	709	59	3400.05	2	2	1	4
8	706	113	5534.97	5	4	2	2
13	709	275	13190.92	2	5	5	4

```
rfm_table['RFM_Score'] = rfm_table['RecencyRank'].astype(str) + rfm_table['FrequencyRank'].astype(str) + rfm_table['MonetaryRank'].
rfm_table.head()
```

household_key	MostRecentDay	Frequency	Monetary	Recency	MonetaryRank	FrequencyRank	RecencyRank	RFM_Score
1	706	86	4330.16	5	3	1	2	213
7	709	59	3400.05	2	2	1	4	412
8	706	113	5534.97	5	4	2	2	224

```
# add RFM_Score to merged table
merged['RFM_Score'] = merged['household_key'].map(rfm_table['RFM_Score'])
merged.head()
```

	household_key	BASKET_ID	DAY	PRODUCT_ID	QUANTITY	SALES_VALUE	STORE_ID	RETAIL_DISC	TRANS_TIME	WEEK_NO	...	DEPARTMENT	CU
0	1364	26984896261	1	842930	1	2.19	31742	0.00	1520	1	...	GROCERY	
1	1364	26984896261	1	897044	1	2.99	31742	-0.40	1520	1	...	GROCERY	
2	1364	26984896261	1	920955	1	3.09	31742	0.00	1520	1	...	MEAT	
3	1364	26984896261	1	937406	1	2.50	31742	-0.99	1520	1	...	MEAT-PCKGD	
4	1364	26984896261	1	981760	1	0.60	31742	-0.79	1520	1	...	GROCERY	

5 rows x 26 columns

4.2 Unit Price

Unit Price

```
# column "Unit Price"
# I am calculating unit price - what the customer actually paid
# opposed to shelf price as per guide - which takes the coupon match instead of coupon discount
# retail discounts are already removed from sales value
# coupon discounts are no, see page 4 of guide
# (sales value - coupon discount)/qty
# coupon discount is already negative, hence add
merged['UNIT_PRICE'] = (merged['SALES_VALUE']+merged['COUPON_DISC'])/merged['QUANTITY']
```

4.3 Redeemed coupon and received loyalty discounts

```
# binary column "Redeemed Coupon"
merged['REDEEMED_COUPON'] = (merged['COUPON_DISC'] < 0).astype(int)
merged['REDEEMED_COUPON'].value_counts()
```

REDEEMED_COUPON	count
0	1402714
1	24589

dtype: int64

```
merged['LOYALTY_DISC'] = (merged['RETAIL_DISC'] < 0).astype(int)
merged['LOYALTY_DISC'].value_counts()
```

LOYALTY_DISC	count
0	714173
1	713130

dtype: int64

4.3.1 Basket size and weekly number of transactions

- Basket size calculated by week and store.
- Weekly number of transactions calculated by week and store.

```
# group merged by week and store, add columns Basket Size, No. of transactions
weekly_basket_size = merged.groupby(['WEEK_NO','STORE_ID'])['QUANTITY'].sum()
weekly_transactions = merged.groupby(['WEEK_NO','STORE_ID'])['BASKET_ID'].nunique()
```

```
weekly_basket_size = weekly_basket_size.reset_index()
#rename column 'QUANTITY' to 'WEEKLY_BASKET_SIZE'
weekly_basket_size.rename(columns={'QUANTITY': 'WEEKLY_BASKET_SIZE'}, inplace=True)
```

```
weekly_transactions = weekly_transactions.reset_index()
#rename column 'BASKET_ID' to 'WEEKLY_TRANSACTIONS'
weekly_transactions.rename(columns={'BASKET_ID': 'WEEKLY_TRANSACTIONS'}, inplace=True)
```

```
weekly_transactions.tail()
```

	WEEK_NO	STORE_ID	WEEKLY_TRANSACTIONS
11484	102	32004	8
11485	102	33923	18
11486	102	34011	20
11487	102	34037	4
11488	102	34280	12

4.4 Categorical gata

```
merged[['COMMODITY_DESC', 'BRAND', 'DEPARTMENT','CURR_SIZE_OF_PRODUCT','display','mailer' ]].nunique()
```

	0
COMMODITY_DESC	305
BRAND	2
DEPARTMENT	41
CURR_SIZE_OF_PRODUCT	3620
display	11
mailer	11

dtype: int64

4.5 Ordinal Encoding

I used ordinal encoding for: AGE_DESC, INCOME_DESC, HOUSEHOLD_SIZE_DESC, KID_CATEGORY_DESC.

```
# ordinal encoding for income_desc
#https://www.geeksforgeeks.org/machine-learning/how-to-perform-ordinal-encoding-using-sklearn/
ordinal_encoder = OrdinalEncoder(categories=[['Under 15K','15-24K','25-34K','35-49K','50-74K','75-99K','100-124K', '125-149K', '150
merged['INCOME_DESC_encoded'] = ordinal_encoder.fit_transform(merged[['INCOME_DESC']])
```

```
# print encoded values
incomes = merged[['INCOME_DESC', 'INCOME_DESC_encoded']].drop_duplicates().sort_values('INCOME_DESC_encoded')
print(incomes)
```

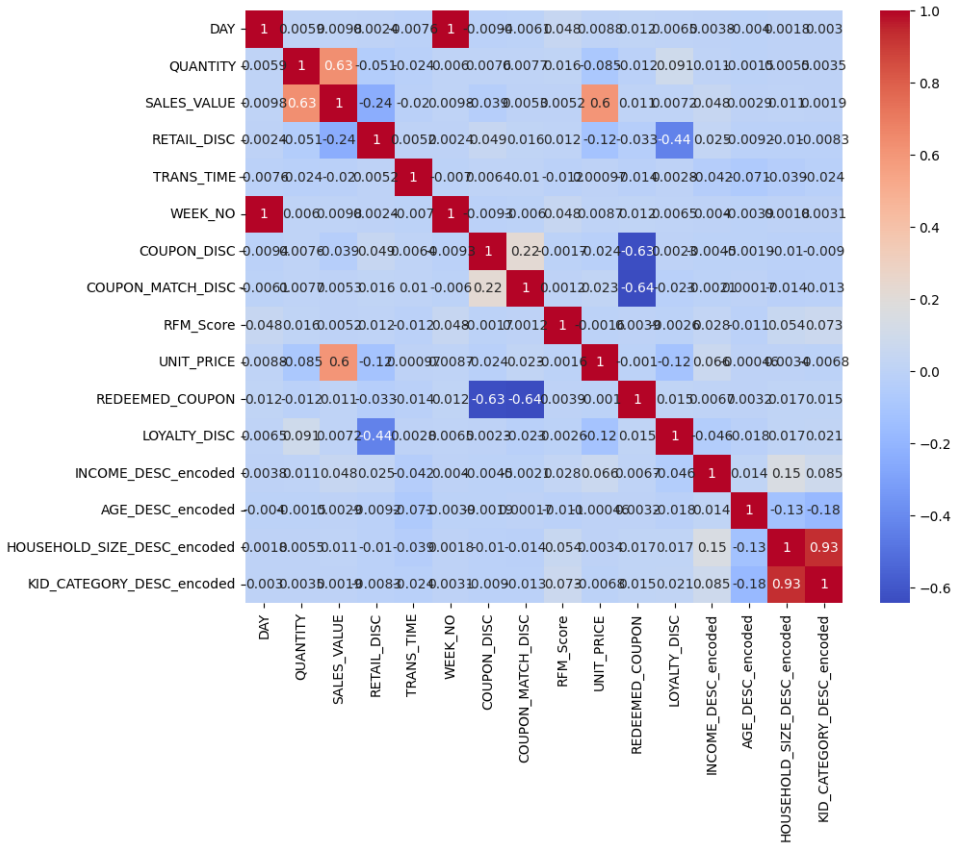
```
      INCOME_DESC  INCOME_DESC_encoded
82     Under 15K                  0.0
433       15-24K                  1.0
5         25-34K                  2.0
10        35-49K                  3.0
15        50-74K                  4.0
191       75-99K                  5.0
0       100-124K                  6.0
290     125-149K                  7.0
1951    150-174K                  8.0
1329    175-199K                  9.0
2612    200-249K                 10.0
734        250K+                 11.0
```

5 EDA

5.1 Correlation matrix

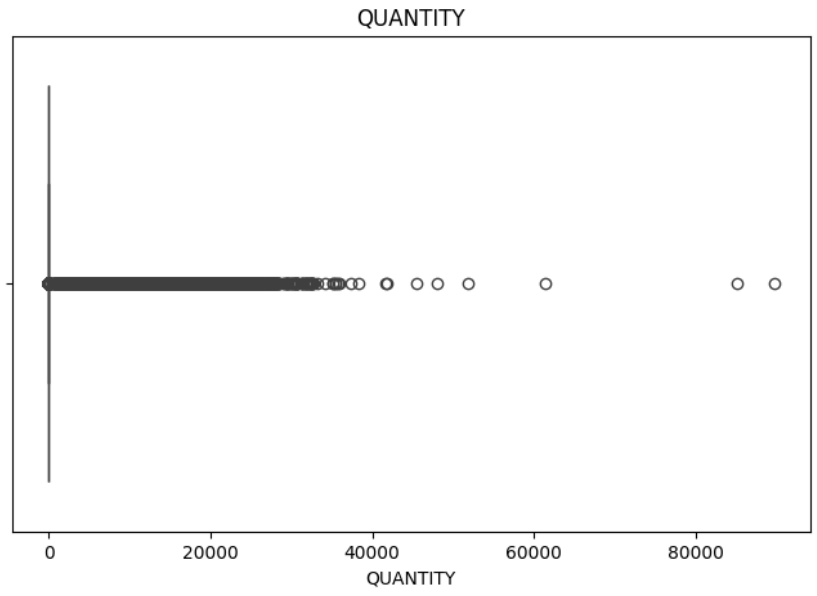
I will not use SALES_VALUE, RETAIL_DISC, COUPON_DISC, COUPON_MATCH_DISC to avoid data leakage, as sales total is derived from units x price. I already created price column and binary for discount and coupons.

```
# correlation matrix excluding BASKET_ID, PRODUCT_ID, STORE_ID
corr_matrix = merged.drop(['household_key','BASKET_ID', 'PRODUCT_ID', 'STORE_ID'], axis=1).corr(numeric_only=True)
corr_matrix
#heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.show()
```



5.2 Quantity boxplot

```
#boxplot for quantity
plt.figure(figsize=(8, 5))
sns.boxplot(x=merged['QUANTITY'],showfliers=True, showcaps=True)
plt.title('QUANTITY')
plt.xlabel('QUANTITY')
plt.show()
```



6 Grouping data for modelling

```
# https://www.geeksforgeeks.org/pandas/python-pandas-dataframe-groupby/
weekly_df = merged.groupby(['WEEK_NO','STORE_ID','household_key','PRODUCT_ID'])\
  .agg(
      Quantity=('QUANTITY', 'sum'),
      SalesValue=('SALES_VALUE', 'sum'),
      display=('display', 'first'),
      mailer=('mailer', 'first'),
      COMMODITY_DESC=('COMMODITY_DESC', 'first'),
      BRAND=('BRAND', 'first'),
      DEPARTMENT=('DEPARTMENT', 'first'),
      CURR_SIZE_OF_PRODUCT=('CURR_SIZE_OF_PRODUCT', 'first'),
      MARITAL_STATUS_CODE =('MARITAL_STATUS_CODE', 'first'),
      HH_COMP_DESC=('HH_COMP_DESC', 'first'),
      HOMEOWNER_DESC=('HOMEOWNER_DESC', 'first'),
      RFM_Score=('RFM_Score', 'first'),
      UNIT_PRICE=('UNIT_PRICE', 'first'),
      REDEEMED_COUPON=('REDEEMED_COUPON', 'first'),
      LOYALTY_DISC=('LOYALTY_DISC', 'first'),
      INCOME_DESC_encoded=('INCOME_DESC_encoded', 'first'),
      AGE_DESC_encoded=('AGE_DESC_encoded', 'first'),
      HOUSEHOLD_SIZE_DESC_encoded=('HOUSEHOLD_SIZE_DESC_encoded', 'first'),
      KID_CATEGORY_DESC_encoded=('KID_CATEGORY_DESC_encoded', 'first')
  )
```

```
weekly_df.reset_index(inplace=True)
weekly_df.head()
```

7 First Model

```
# train, validation (4 weeks) and test (4 weeks)
train = df.loc[(df['WEEK_NO']<95), :]
val = df.loc[(df['WEEK_NO']>=95) & (df['WEEK_NO']<99), :]
test = df.loc[(df['WEEK_NO']>=99), :]
```

```
# define target and input features, split info train, validation, test
y_train = train['Quantity']
y_val = val['Quantity']
y_test = test['Quantity']

features = ['WEEK_NO', 'BRAND', 'DEPARTMENT','CURR_SIZE_OF_PRODUCT','RFM_Score',
            'UNIT_PRICE', 'LOYALTY_DISC', 'WEEKLY_BASKET_SIZE',
            'WEEKLY_TRANSACTIONS']
X_train = train[features]
X_val = val[features]
X_test = test[features]

# check the shapes of each train, val, test
y_train.shape, y_val.shape, y_test.shape, X_train.shape, X_val.shape, X_test.shape
```

```
((1262144,), (62875,), (58729,), (1262144, 9), (62875, 9), (58729, 9))
```

```
# initialise XGBoost regressor, enable native categorical handling
xgb_reg = xgb.XGBRegressor(objective = 'reg:squarederror', n_estimators=1000,
                           tree_method = 'hist',enable_categorical = True,
                           max_depth=7, eta=0.01, subsample=0.7, colsample_bytree=0.8,
                           random_state=42
                           )

# fit the model with validation set
# frequency of logging training progress to the console: 100 iterations
# I could not use early stopping with callbacks
xgb_reg.fit(X_train, y_train,
            eval_set=[(X_val, y_val)],
            verbose=100)
```

```python
# make predictions on test
y_pred = xgb_reg.predict(X_test)

# evaluate the model: RMSE, MAE, MSE, R2
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print("RMSE:", rmse)
print("MAE:", mae)
print("MSE:", mse)
print("R2:", r2)
```

```
RMSE: 549.2253635803795
MAE: 38.51579666137695
MSE: 301648.5
R2: 0.8442577719688416
```

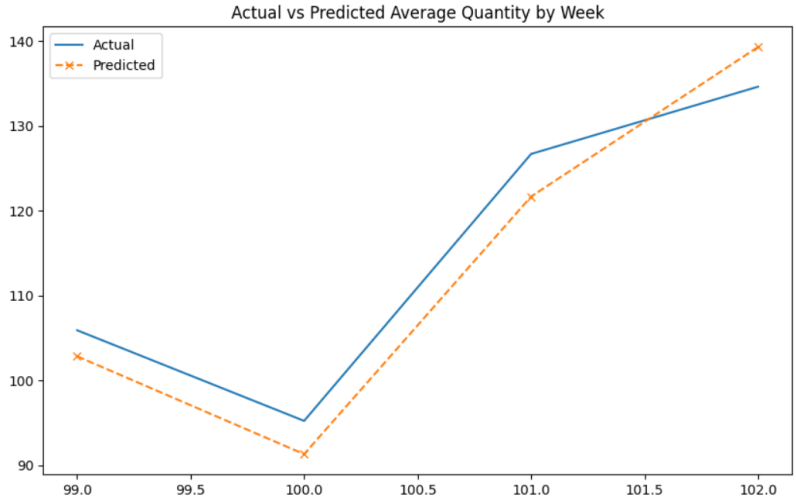
```python
# Make predictions on train and validation sets
y_train_pred = xgb_reg.predict(X_train)
y_val_pred = xgb_reg.predict(X_val)

# Evaluate Training Set
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
mae_train = mean_absolute_error(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("Training Set Metrics")
print("RMSE: ", rmse_train)
print("MAE: ", mae_train)
print("MSE: ", mse_train)
print("R2: ", r2_train)

# Evaluate Validation Set
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))
mae_val = mean_absolute_error(y_val, y_val_pred)
mse_val = mean_squared_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)

print("\n Validation Set Metrics")
print("RMSE: ", rmse_val)
print("MAE: ", mae_val)
print("MSE: ", mse_val)
print("R2: ", r2_val)
```

7.1 Feature importance (most used featured)

```
importance_df = pd.DataFrame({'feature': X_train.columns,
                              'importance': xgb_reg.feature_importances_})
importance_df = importance_df.sort_values('importance', ascending=False)
importance_df
```

	feature	importance
2	DEPARTMENT	0.532514
6	LOYALTY_DISC	0.162804
5	UNIT_PRICE	0.155599
3	CURR_SIZE_OF_PRODUCT	0.054468
1	BRAND	0.041390
7	WEEKLY_BASKET_SIZE	0.017948
4	RFM_Score	0.017749
8	WEEKLY_TRANSACTIONS	0.009785
0	WEEK_NO	0.007743

Actual vs Predicted



8 Second Model

```
# train, validation (4 weeks) and test (4 weeks)
train = df.loc[(df['WEEK_NO']<95), :]
val = df.loc[(df['WEEK_NO']>=95) & (df['WEEK_NO']<99), :]
test = df.loc[(df['WEEK_NO']>=99), :]
```

```
# define target and input features, split info train, validation, test
y_train = train['Quantity']
y_val = val['Quantity']
y_test = test['Quantity']

features = ['WEEK_NO', 'BRAND', 'DEPARTMENT','CURR_SIZE_OF_PRODUCT','RFM_Score',
            'UNIT_PRICE', 'LOYALTY_DISC', 'display','mailer','MARITAL_STATUS_CODE',
            'HH_COMP_DESC', 'HOMEOWNER_DESC','REDEEMED_COUPON','INCOME_DESC_encoded',
            'AGE_DESC_encoded','HOUSEHOLD_SIZE_DESC_encoded','KID_CATEGORY_DESC_encoded',
            'WEEKLY_BASKET_SIZE', 'WEEKLY_TRANSACTIONS']
X_train = train[features]
X_val = val[features]
X_test = test[features]

# check the shapes of each train, val, test
y_train.shape, y_val.shape, y_test.shape, X_train.shape, X_val.shape, X_test.shape
```

```
((1262144,), (62875,), (58729,), (1262144, 19), (62875, 19), (58729, 19))
```

```
# initialise XGBoost regressor, enable native categorical handling
xgb_reg = xgb.XGBRegressor(objective = 'reg:squarederror', n_estimators=1000,
                           tree_method = 'hist',enable_categorical = True,
                           max_depth=7, eta=0.01, subsample=0.7, colsample_bytree=0.8,
                           random_state=42
                           )

# fit the model with validation set
# frequency of logging training progress to the console: 100 iterations
# I could not use early stopping with callbacks
xgb_reg.fit(X_train, y_train,
            eval_set=[(X_val, y_val)],
            verbose=100)
```

```
# make predictions on test
y_pred = xgb_reg.predict(X_test)

# evaluate the model: RMSE, MAE, MSE, R2
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
mape = np.mean(np.abs((y_test - y_pred) / y_test)) * 100

print("RMSE:", rmse)
print("MAE:", mae)
print("MSE:", mse)
print("R2:", r2)
```

```
# Make predictions on train and validation sets
y_train_pred = xgb_reg.predict(X_train)
y_val_pred = xgb_reg.predict(X_val)

# Evaluate Training Set
rmse_train = np.sqrt(mean_squared_error(y_train, y_train_pred))
mae_train = mean_absolute_error(y_train, y_train_pred)
mse_train = mean_squared_error(y_train, y_train_pred)
r2_train = r2_score(y_train, y_train_pred)

print("Training Set Metrics")
print("RMSE: ", rmse_train)
print("MAE: ", mae_train)
print("MSE: ", mse_train)
print("R2: ", r2_train)

# Evaluate Validation Set
rmse_val = np.sqrt(mean_squared_error(y_val, y_val_pred))
mae_val = mean_absolute_error(y_val, y_val_pred)
mse_val = mean_squared_error(y_val, y_val_pred)
r2_val = r2_score(y_val, y_val_pred)

print("\n Validation Set Metrics")
print("RMSE: ", rmse_val)
print("MAE: ", mae_val)
print("MSE: ", mse_val)
print("R2: ", r2_val)
```

8.1 Feature importance (most used feature)

property feature_importances_ --> Feature importances property

```
importance_df = pd.DataFrame({'feature': X_train.columns,
                              'importance': xgb_reg.feature_importances_})
importance_df = importance_df.sort_values('importance', ascending=False)
importance_df
```

```
# plot feature importance in horizontal bar
plt.figure(figsize=(10, 6))

#change from highest to lowest
importance_df_sorted = importance_df.sort_values('importance', ascending=True)

plt.barh(importance_df_sorted['feature'], importance_df_sorted['importance'])
# plt.xlabel('Importance')
plt.title('Feature Importance')
#plt.tight_layout()
plt.show()
```

8.2 Actual vs Predicted

```
# plot actual vs predicted average quantity by week
# group average quantity by week
weekly_actual = y_test.groupby(X_test['WEEK_NO']).mean()

# mapping predictions to their week numbers
predicted_data_for_plot = pd.DataFrame({
    'WEEK_NO': X_test['WEEK_NO'].values,
    'Predicted_Quantity': y_pred
})
weekly_predicted = predicted_data_for_plot.groupby('WEEK_NO')['Predicted_Quantity'].mean()

# line chart
plt.figure(figsize=(10, 6))
plt.plot(weekly_actual.index, weekly_actual.values, label='Actual')
plt.plot(weekly_predicted.index, weekly_predicted.values, label='Predicted',marker='x', linestyle='--')
#plt.ylabel('Average Quantity')
#plt.xlabel('Week')
plt.title('Actual vs Predicted Average Quantity by Week')
plt.legend()
plt.show()
```

8.3 SHAP

```
# SHAP explainer
explainer = shap.Explainer(xgb_reg)
```

```
# SHAP values for the test set
shap_values = explainer(X_test)
```

```
# visualise values
shap.summary_plot(shap_values, X_test)
```

```
# mean shap values
mean_shap_values = np.abs(shap_values.values).mean(axis=0)
```

```
feature_col = X_test.columns
# sort
sorted_features = np.argsort(mean_shap_values)
```

```
# Plot of mean SHAP
plt.figure(figsize=(10, 6))
plt.barh(feature_col[sorted_features], mean_shap_values[sorted_features])
plt.xlabel('Mean SHAP Value')
plt.title('Mean SHAP Value for Each Feature')
plt.show()
```