Configuration Manual

MSc Research Project
Master of Science in Data Analytics

Shailesh Subhashchand Yadav
Student ID: X21222801

School of Computing
National College of Ireland

Supervisor: Dr Syed Muslim Jameel

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shailesh Subhashchand Yadav
Student ID:	x21222801
Programme:	MSc Data Analytics **Year:** 2022-23
Module:	MSc Research Project
Lecturer:	Dr Syed Muslim Jameel
Submission Due Date:	18-09-2023
Project Title:	Sales Prediction for Small and Medium Enterprises using Machine Learning
Word Count:	1142 **Page Count: 20**

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: Shailesh Subhashchand Yadav

Date: 18th September 2023

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 of Sales Prediction for Small Medium Enterprises Using Machine Learning

Shailesh Yadav
Student ID: x21222801

1 Introduction

The prerequisites required for performing our research smoothly will all be explained in this document. This configuration manual document will not only explain all the software as well as hardware tools used but also will talk in detail about all the steps taken for completion of our research of sales prediction for small medium enterprises using machine learning.

2 System Configuration

The software and hardware tools used to perform this research are listed below.

2.1 Hardware Configuration

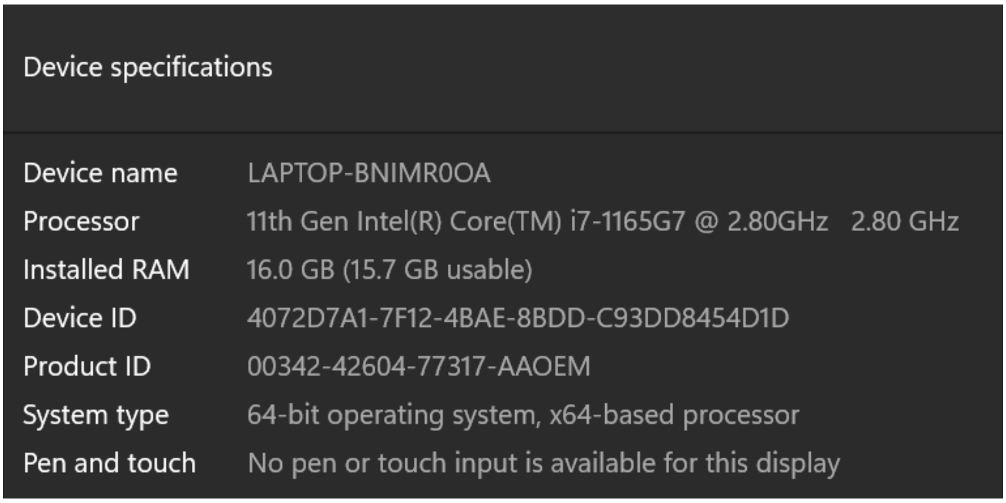

Fig 1 :- Hardware specification

- Fig 1 showcases the device specification where we see the 11th gen i7 processor , the system contains 16 GB of RAM and 500 GB of SSD memory.

2.2 Sofware Configuration

- Fig 2 Showcases the OS build number and the operating system version which is 22H2

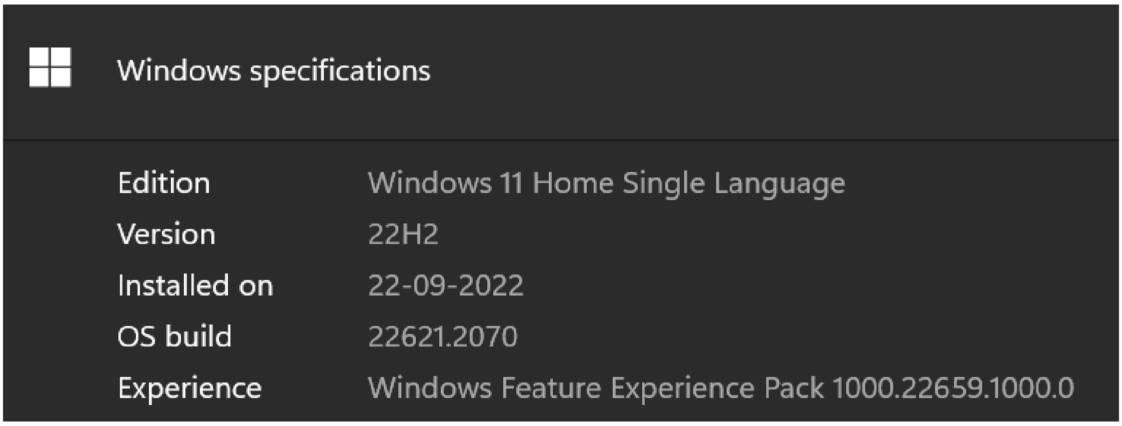

Fig 2. System OS specification

- Python version

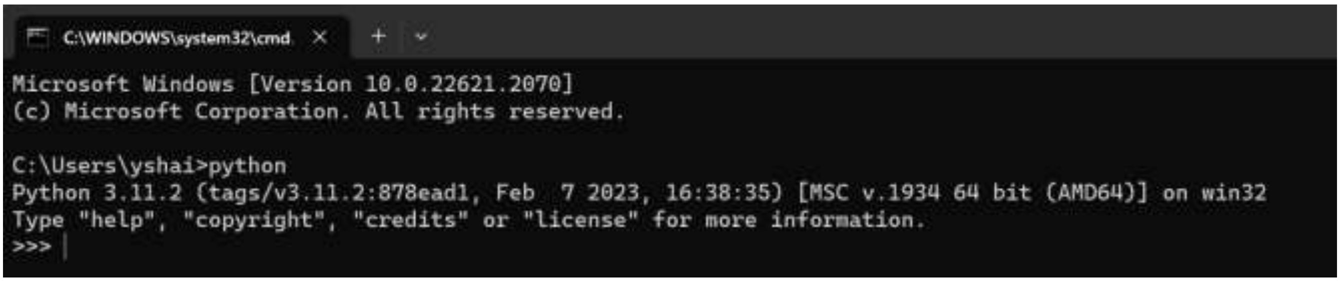

Fig 3 :- Python Version

- MS Excel

Fig 4 :- MS Excel Version

- **Google Colab**

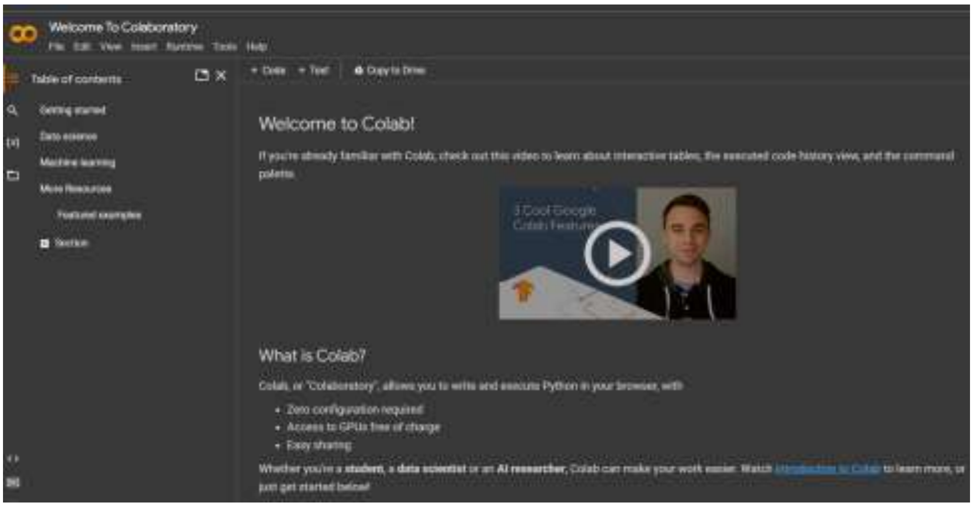

Fig 5:- Google Colab Configuration

3 Implementations and Results

3.1 Dataset

- Both the datasets are being downloaded from Kaggle
- Kaggle is an online website where publicly free datasets are available

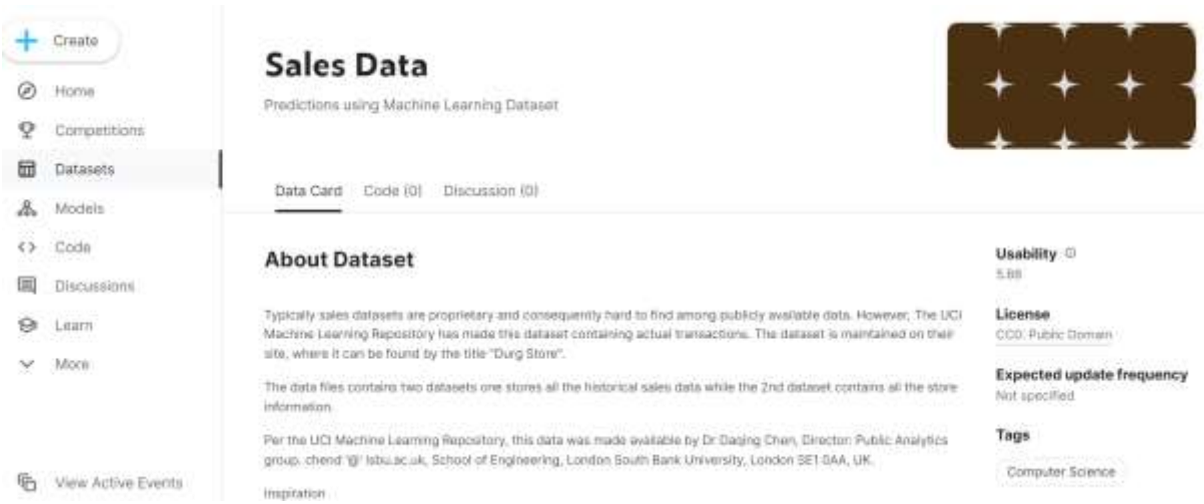

Fig 6:- Medium Enterprises Dataset

The medium enterprises dataset is huge and sales_df contains 1017208 rows and 9 columns and store_df contains 1116 rows and 10 columns

- Small Enterprises Dataset which was taken from Kaggle

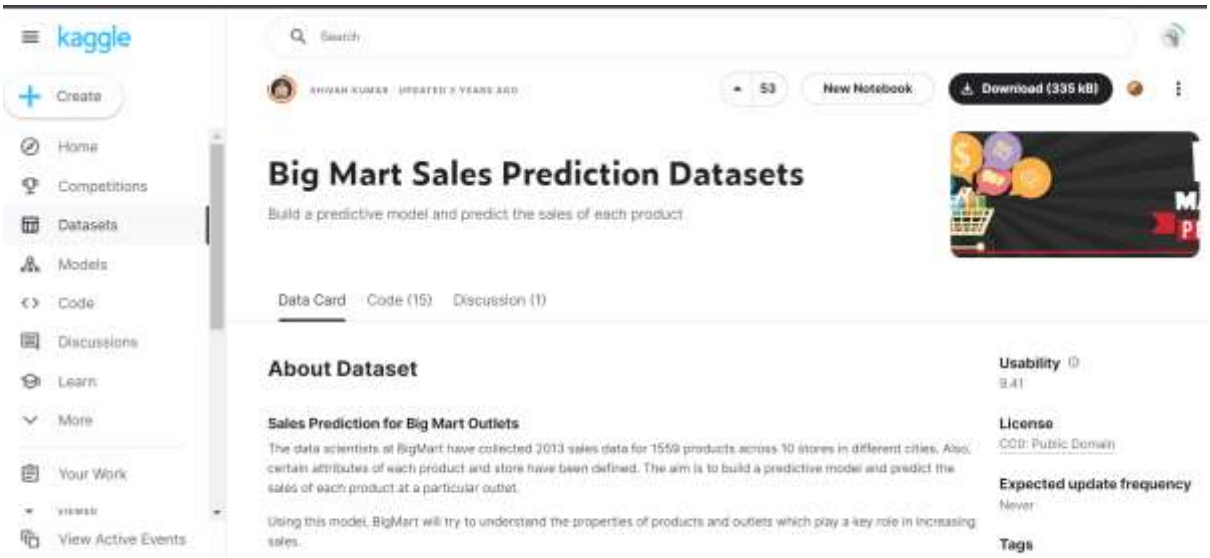

Fig 7:- Small Enterprises Dataset

The small Enterprises Dataset Contains train data of 8523 rows and test data of 5681 rows , the train data set has both input and output variable.

3.2 Installing Required Packages

- Importing all then required packages required for the research

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
from datetime import datetime
import seaborn as sns
import plotly.express as px
import pandas as pd
import ast
import math
import random
from scipy.stats import skew
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
import xgboost as xgb
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

Fig 8:- Python Packages to be installed and imported

3.3 Dataset loading on Google Colab

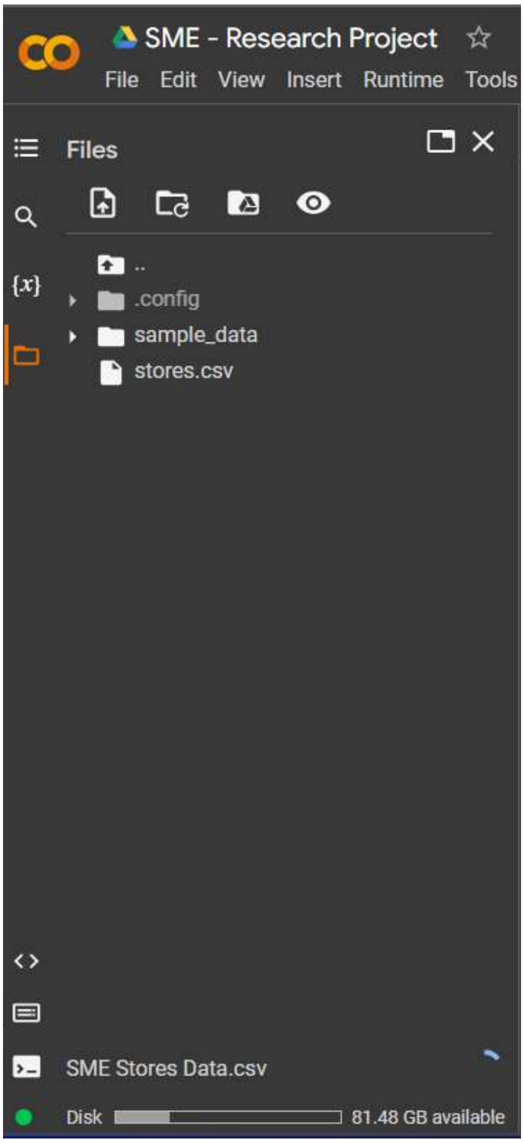

Fig 9:- Uploading the Medium enterprises dataset on Google Colab

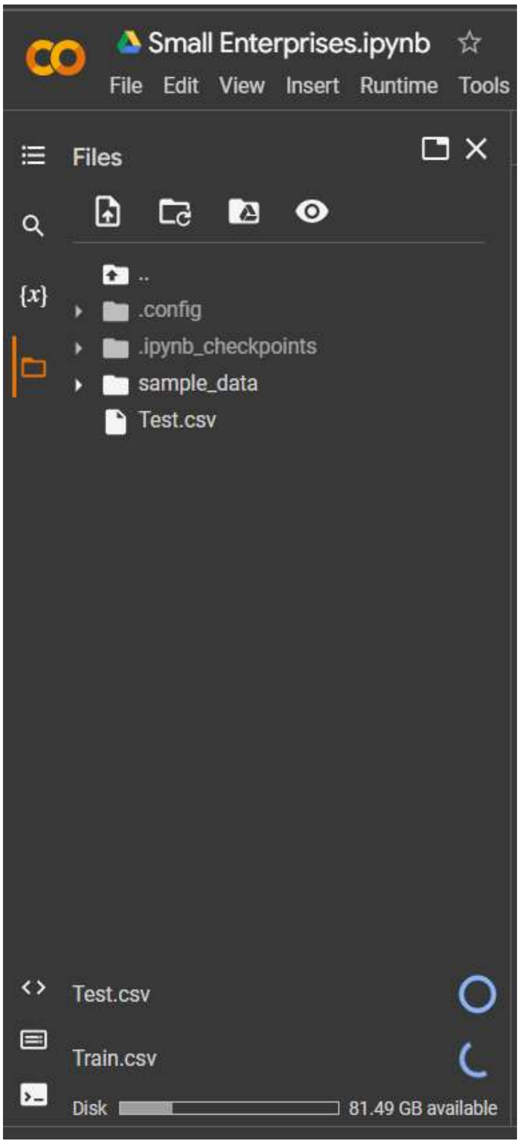

Fig 9.(a):- Uploading the Small Enterprises Data

- The dataset are downloaded from Kaggle and now the dataset are been uploaded into Google colab.
- We can see in Fig 9 that store.csv has been uploaded completely whereas SME Stores Data.csv is still being uploaded.
- In Fig 9(a) we see that both the test.csv and train.csv are been uploaded into Google Colab database.

3.4 Loading the Dataset

```
Dataset Loading

#SME Data
database = "/content/SME Stores Data.csv"
sales_df =pd.read_csv(database)
# Store data
database = "/content/stores.csv"
store_df =pd.read_csv(database)

<ipython-input-9-7da53c7fdee1>:3: DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or set low_memory=False.
  sales_df =pd.read_csv(database)
```

Fig 10:- Dataset loading for Medium Enterprises

```
#SmallEnt Data
database = "/content/Train.csv"
sales_df =pd.read_csv(database)
# Store data
database = "/content/Test.csv"
store_df =pd.read_csv(database)
```

Fig 11:- Dataset loading for Small Medium Enterprises

- Fig 10 and 11 shows the data being imported into google colab for our research

3.5 Data Pre-Processing

- Checking Null Values

```
sales_df.isnull().sum()

Store            0
DayOfWeek        0
Date             0
Sales            1
Customers        1
Open             1
Promo            1
StateHoliday     1
SchoolHoliday    1
dtype: int64

store_df.isnull().sum()

Store                          0
StoreType                      0
Assortment                     0
CompetitionDistance            3
CompetitionOpenSinceMonth    354
CompetitionOpenSinceYear     354
Promo2                         0
Promo2SinceWeek              544
Promo2SinceYear              544
PromoInterval                544
dtype: int64
```

```
sales_df.isnull().sum()

Store            0
DayOfWeek        0
Date             0
Sales            0
Customers        0
Open             0
Promo            0
StateHoliday     0
SchoolHoliday    0
dtype: int64

store_df.isna().sum()

Store                        0
StoreType                    0
Assortment                   0
CompetitionDistance          0
CompetitionOpenSinceMonth    0
CompetitionOpenSinceYear     0
Promo2                       0
Promo2SinceWeek              0
Promo2SinceYear              0
PromoInterval                0
dtype: int64
```

Fig 12 :- Cleaning null values in Medium Enterprises dataset

- We can see that in sales_df there are 6 rows having null values and in store_df 5 rows are having null values so we clean this null values show in fig 12

- Null values in small Enterprises Dataset

```
sales_df.isnull().apply(lambda x: (x.sum(), x.sum()/sales_df.shape[0] *100)).transpose()
```

	0	1
Item_Identifier	0.0	0.000000
Item_Weight	1463.0	17.165317
Item_Fat_Content	0.0	0.000000
Item_Visibility	0.0	0.000000
Item_Type	0.0	0.000000
Item_MRP	0.0	0.000000
Outlet_Identifier	0.0	0.000000
Outlet_Establishment_Year	0.0	0.000000
Outlet_Size	2410.0	28.276428
Outlet_Location_Type	0.0	0.000000
Outlet_Type	0.0	0.000000
Item_Outlet_Sales	0.0	0.000000

- We have 17.16 % and 28.27 % of missing values in Item weight and Outlet_Size columns respectively in Train Datasets.

```
store_df.isnull().apply(lambda x: (x.sum(), x.sum()/store_df.shape[0] *100)).transpose()
```

	0	1
Item_Identifier	0.0	0.000000
Item_Weight	976.0	17.180074
Item_Fat_Content	0.0	0.000000
Item_Visibility	0.0	0.000000
Item_Type	0.0	0.000000
Item_MRP	0.0	0.000000
Outlet_Identifier	0.0	0.000000
Outlet_Establishment_Year	0.0	0.000000
Outlet_Size	1606.0	28.269671
Outlet_Location_Type	0.0	0.000000
Outlet_Type	0.0	0.000000

- We have 17% and 28% of missing values in Item weight and Outlet_Size columns in Test Dataset, i.e. around same of sales_df dataset.

Fig 13 :- Null Values in Small Enterprises Dataset

- Checking Datatypes

```
store_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1115 entries, 0 to 1114
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Store                      1115 non-null   int64
 1   StoreType                  1115 non-null   object
 2   Assortment                 1115 non-null   object
 3   CompetitionDistance        1112 non-null   float64
 4   CompetitionOpenSinceMonth  761 non-null    float64
 5   CompetitionOpenSinceYear   761 non-null    float64
 6   Promo2                     1115 non-null   int64
 7   Promo2SinceWeek            571 non-null    float64
 8   Promo2SinceYear            571 non-null    float64
 9   PromoInterval              571 non-null    object
dtypes: float64(5), int64(2), object(3)
memory usage: 87.2+ KB
```

```
sales_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1017209 entries, 0 to 1017208
Data columns (total 9 columns):
 #   Column         Non-Null Count    Dtype
---  ------         --------------    -----
 0   Store          1017209 non-null  int64
 1   DayOfWeek      1017209 non-null  int64
 2   Date           1017209 non-null  object
 3   Sales          1017209 non-null  int64
 4   Customers      1017209 non-null  int64
 5   Open           1017209 non-null  int64
 6   Promo          1017209 non-null  int64
 7   StateHoliday   1017209 non-null  object
 8   SchoolHoliday  1017209 non-null  int64
dtypes: int64(7), object(2)
memory usage: 69.8+ MB
```

```
sales_df.dtypes
```

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content              object
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                   object
Outlet_Location_Type          object
Outlet_Type                   object
Item_Outlet_Sales            float64
dtype: object
```

```
store_df.dtypes
```

```
Item_Identifier               object
Item_Weight                  float64
Item_Fat_Content              object
Item_Visibility              float64
Item_Type                     object
Item_MRP                     float64
Outlet_Identifier             object
Outlet_Establishment_Year      int64
Outlet_Size                   object
Outlet_Location_Type          object
Outlet_Type                   object
dtype: object
```

Fig 14 :- Datatypes of both SME

- Replacing Null values with 0 in Medium enterprises

```
# STORE DATASET FILL INTO NULL VALUES I.E 0
store_df['CompetitionDistance'] = store_df['CompetitionDistance'].fillna(0)
store_df['CompetitionOpenSinceMonth'] = store_df['CompetitionOpenSinceMonth'].fillna(0)
store_df['CompetitionOpenSinceYear'] = store_df['CompetitionOpenSinceYear'].fillna(0)
store_df['Promo2SinceWeek'] = store_df['Promo2SinceWeek'].fillna(0)
store_df['Promo2SinceYear'] = store_df['Promo2SinceYear'].fillna(0)
store_df['PromoInterval'] = store_df['PromoInterval'].fillna(0)
```

Fig 15 :- Null replacment into 0

- Checking Outliers in Small Enterprises

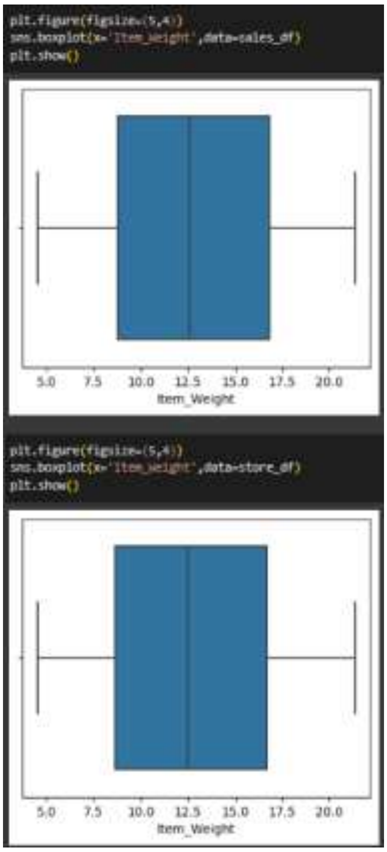

Fig 16 : Checking outliers

- In fig 16 we can seee that there are no outliers in the dataset

- Data Imputation in Small Enterprises Dataset

```
sales_df['Outlet_Size'].isnull().sum(),store_df['Outlet_Size'].isnull().sum()

(2410, 1606)

sales_df['Outlet_Size'].value_counts()

Medium    2793
Small     2388
High       932
Name: Outlet_Size, dtype: int64

store_df['Outlet_Size'].value_counts()

Medium    1862
Small     1592
High       621
Name: Outlet_Size, dtype: int64
```

- Since the outlet_size is a categorical column, we can impute the missing values by "Mode"(Most Repeated Value) from the column.

```
sales_df['Outlet_Size']= sales_df['Outlet_Size'].fillna(sales_df['Outlet_Size'].mode()[0])
store_df['Outlet_Size']= store_df['Outlet_Size'].fillna(store_df['Outlet_Size'].mode()[0])

sales_df['Outlet_Size'].isnull().sum(),store_df['Outlet_Size'].isnull().sum()

(0, 0)
```

- We have succesfully imputed the missing values from the column Outlet_Size

Fig 17 :- Data imputation in Outlet_Size column.

- Data Merger of Medium Enterprises Dataset

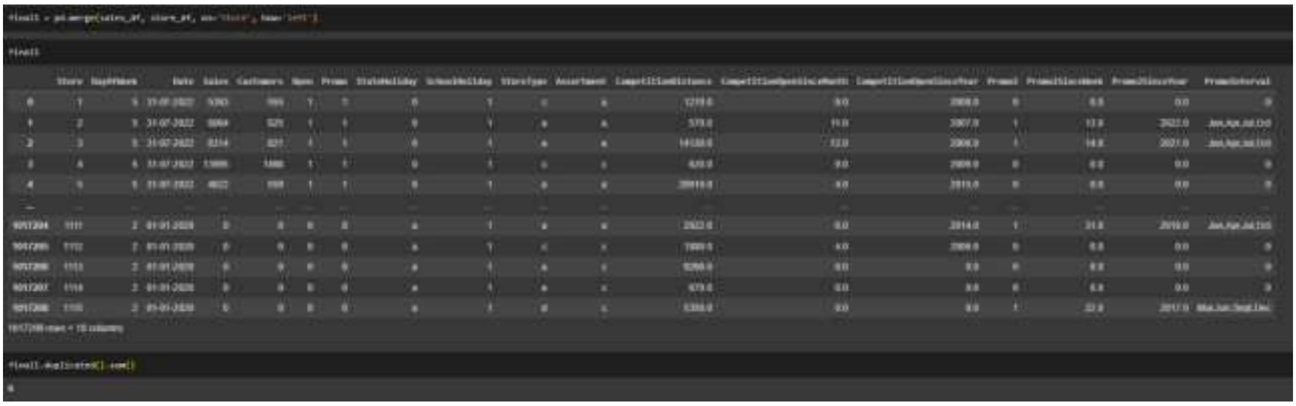

Fig 18:- Data Merger

- Here we have merged both the dataset together and checked for any duplicat values which is found to be 0 , so there are no duplicates value in the merged dataset.

- Changing the Datatypes

```python
#Change data types object to int
final1.loc[final1['StateHoliday'] == '0', 'StateHoliday'] = 0
final1.loc[final1['StateHoliday'] == 'a', 'StateHoliday'] = 1
final1.loc[final1['StateHoliday'] == 'b', 'StateHoliday'] = 2
final1.loc[final1['StateHoliday'] == 'c', 'StateHoliday'] = 3
#store the value with same column name i.e StateHoliday with function astype
final1['StateHoliday'] = final1['StateHoliday'].astype(int, copy=False)
```

```python
# change Data Types object into int
final1.loc[final1['Assortment'] == 'a', 'Assortment'] = 0
final1.loc[final1['Assortment'] == 'b', 'Assortment'] = 1
final1.loc[final1['Assortment'] == 'c', 'Assortment'] = 2
#store the value with same column name i.e Assortment with function astype
final1['Assortment'] = final1['Assortment'].astype(int, copy=False)
```

```python
# change Data Types object into int
final1.loc[final1['StoreType'] == 'a', 'StoreType'] = 0
final1.loc[final1['StoreType'] == 'b', 'StoreType'] = 1
final1.loc[final1['StoreType'] == 'c', 'StoreType'] = 2
final1.loc[final1['StoreType'] == 'd', 'StoreType'] = 3
#store the value with same column name i.e Assortment with function astype
final1['StoreType'] = final1['StoreType'].astype(int, copy=False)
```

```
final1.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  object
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  int64
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  int64
 10  Assortment                 1017209 non-null  int64
 11  CompetitionDistance        1017209 non-null  float64
 12  CompetitionOpenSinceMonth  1017209 non-null  float64
 13  CompetitionOpenSinceYear   1017209 non-null  float64
 14  Promo2                     1017209 non-null  int64
 15  Promo2SinceWeek            1017209 non-null  float64
 16  Promo2SinceYear            1017209 non-null  float64
 17  PromoInterval              1017209 non-null  object
dtypes: float64(5), int64(11), object(2)
memory usage: 147.5+ MB
```

Fig 19:- Datatypes Changes

- Here in Fig 19 we see that the merged dataset is being checked and object datatypes are converted into int.

- Data Transformation

```
# code for changing format of date from object to datetime
final1['Date'] = pd.to_datetime(final1['Date'], format= '%d-%m-%Y')
```

```
# code for change object into date format
final1['CompetitionOpenSinceMonth'] = pd.DatetimeIndex(final1['Date']).month
```

```
# code for change float into integer
final1['CompetitionOpenSinceYear']= final1['CompetitionOpenSinceYear'].astype(int)
final1['Promo2SinceYear']= final1['Promo2SinceYear'].astype(int)
```

```
# code for change float into integer
final1['CompetitionDistance']= final1['CompetitionDistance'].astype(int)
final1['Promo2SinceWeek']= final1['Promo2SinceWeek'].astype(int)
```

```
final1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1017209 entries, 0 to 1017208
Data columns (total 18 columns):
 #   Column                     Non-Null Count    Dtype
---  ------                     --------------    -----
 0   Store                      1017209 non-null  int64
 1   DayOfWeek                  1017209 non-null  int64
 2   Date                       1017209 non-null  datetime64[ns]
 3   Sales                      1017209 non-null  int64
 4   Customers                  1017209 non-null  int64
 5   Open                       1017209 non-null  int64
 6   Promo                      1017209 non-null  int64
 7   StateHoliday               1017209 non-null  int64
 8   SchoolHoliday              1017209 non-null  int64
 9   StoreType                  1017209 non-null  int64
 10  Assortment                 1017209 non-null  int64
 11  CompetitionDistance        1017209 non-null  int64
 12  CompetitionOpenSinceMonth  1017209 non-null  int64
```

Fig 20 :- Data transformation

- Here in fig 20, we see all the float vales are converted into int64 and date object is also been converted into datetime format.

- Exploratory Data Analysis (EDA) for Medium Enterprises Dataset

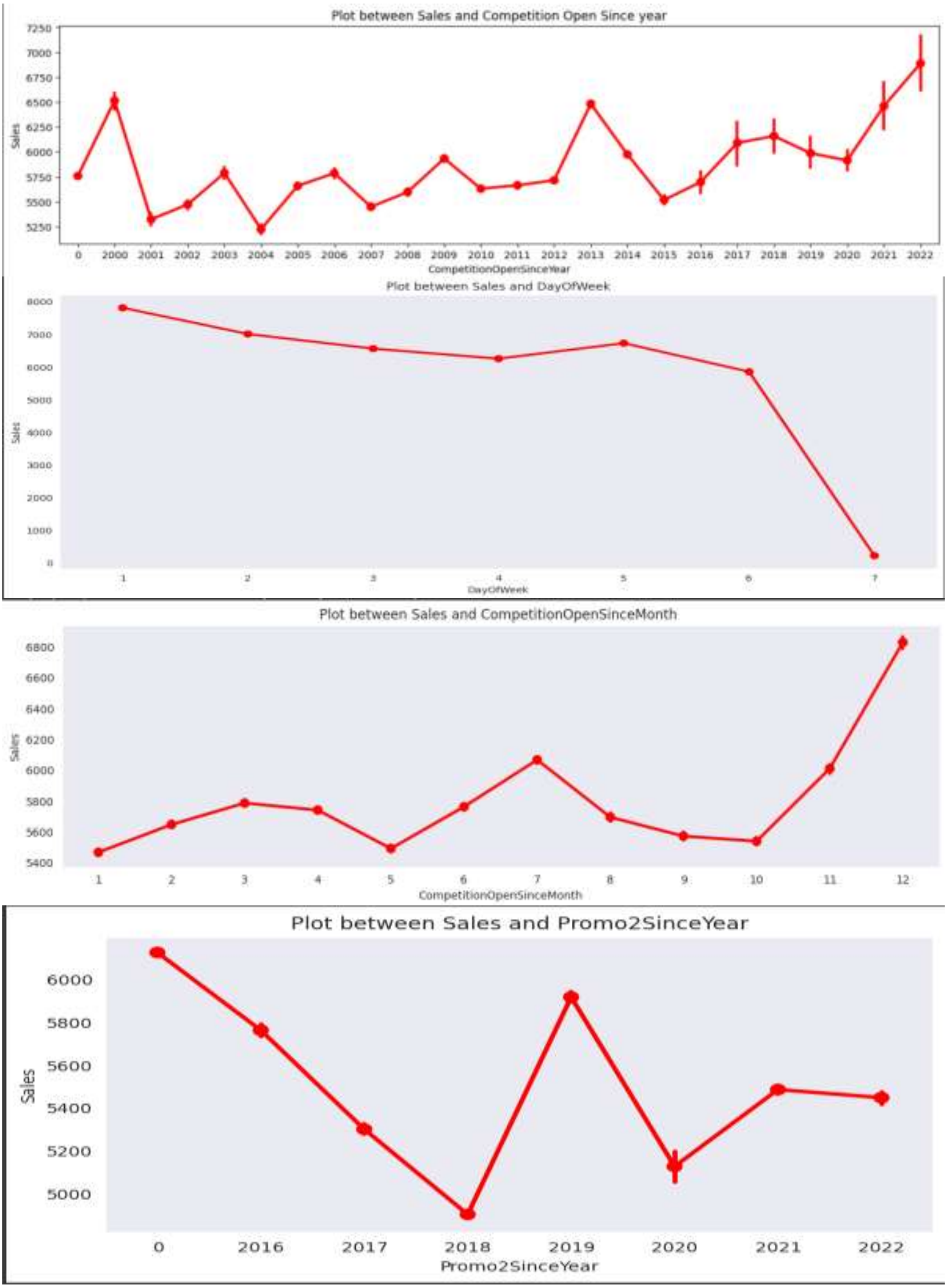

Fig 21:- EDA of Medium Enterprises Dataset

EDA for Small Enterprise Dataset

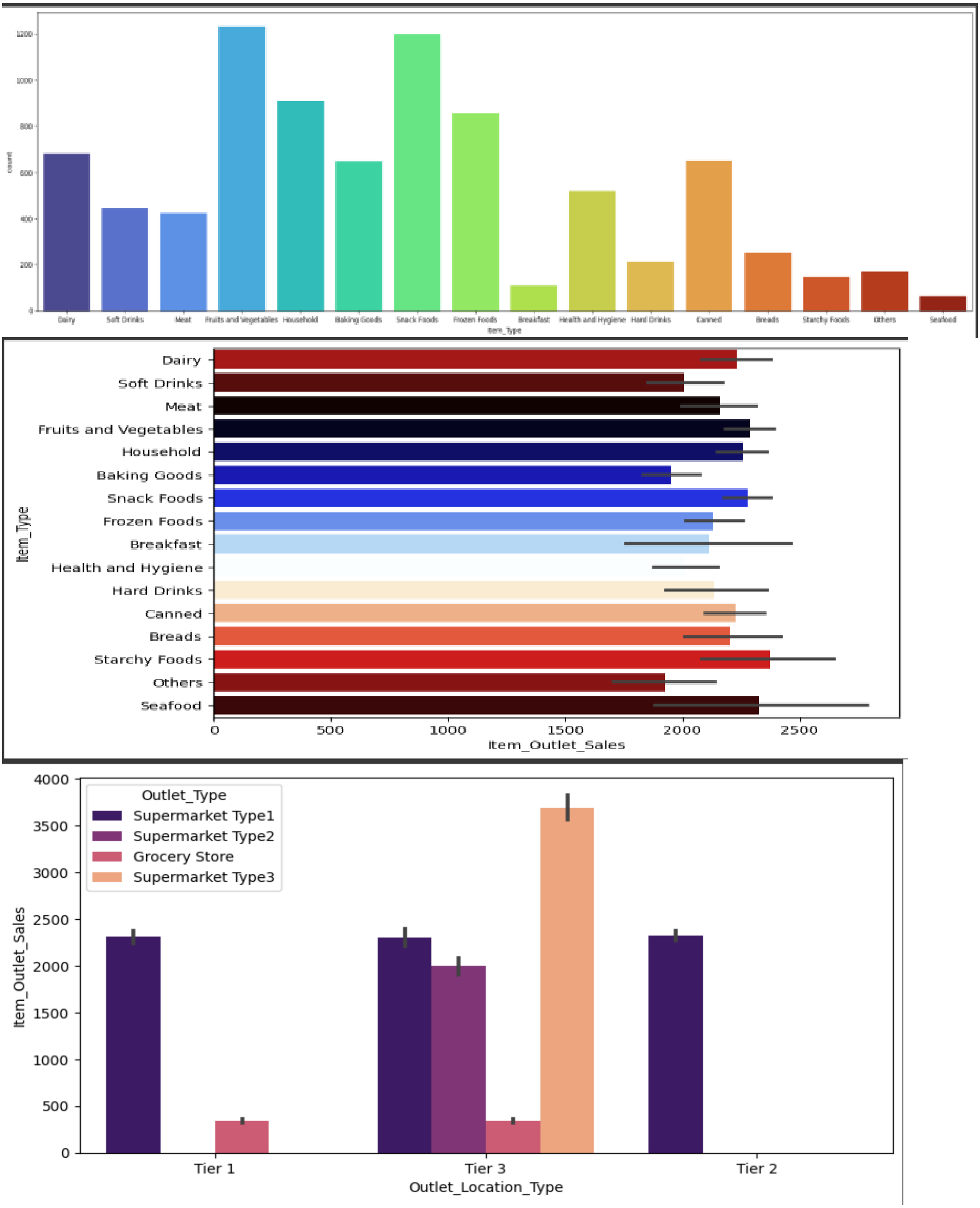

Fig 22: EDA of Small Enterprise Dataset

- Feature Selection

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

    return(vif)
```

```
[ ]  calc_vif(final1[[i for i in final1.describe().columns if i not in ['Sales']]])
```

Fig 23:- Multicollinearity

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

```
standard_Item_Weight = StandardScaler()
standard_Item_Weight.fit(X[['Item_Weight']])

X['Item_Weight'] =  standard_Item_Weight.transform(X[['Item_Weight']])
sns.displot(X.Item_Weight, kde=True)
plt.show()
```

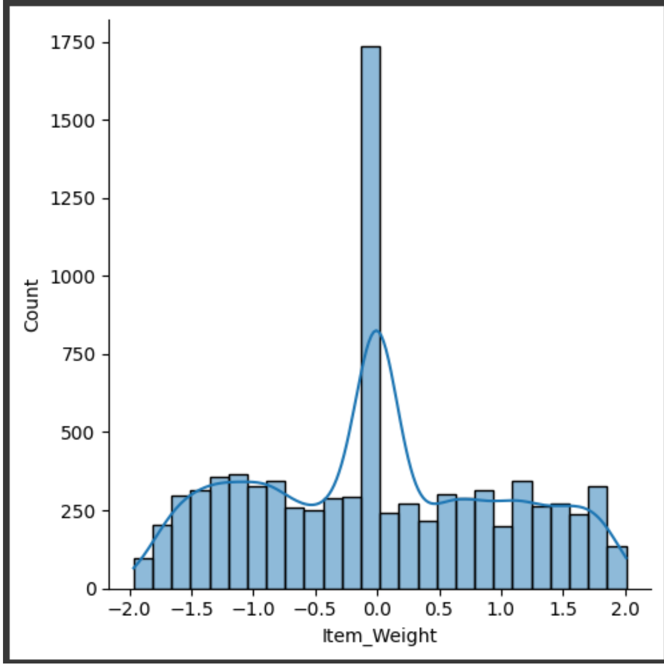

Fig 24 : Standardization

- Model Building

Linear Regression

```
[ ] # Here we Train the model
    reg = LinearRegression().fit(X_train, y_train)

    #Checking the Regression Score i.e R-squared value
    reg.score(X_train, y_train)

    0.7635177898918

[ ] # Checking the cofficient of different independent columns
    reg.coef_

    array([-1.61211918e-01, -6.85555790e+01,  6.35502790e+00, -4.63966643e-11,
            1.29046381e+03, -1.28103146e+03,  3.62299716e+01,  2.76744328e+02,
            2.22203882e+02,  2.88842934e-02,  2.99492517e+01,  8.38855419e-02,
           -4.31766708e+01,  1.24869391e+01,  4.31766708e+01,  3.17457053e+01,
            1.38824067e+02, -2.13746443e+02])

[ ] # Checking the intercept of different Indpendent columns
    reg.intercept_

    700.3663214004991

[ ] #Predicting Dependent Variable With Test Dataset i.e 20%
    y_pred = reg.predict(X_test)
    y_pred

    array([5324.46083545, 4849.36639321, 7238.30961726, ..., 4090.08404244,
           7772.91618311, 5368.30274156])

[ ] # Original Test Dependent Value
    y_test

[ ] #Predicting on Train Dataset
    y_pred_train = reg.predict(X_train)
    y_pred_train

    array([5400.55393279, 6804.33388843, 6699.08503089, ..., 7913.63233424,
           6884.83997692, 6122.83768554])

[ ] # Dependent Variable With Train Dataset i.e 80 %
    y_train

    array([5400, 7339, 7887, ..., 4817, 6194, 5862])

[ ] #Calculate MSE & RMSE for Test Prediction
    MSE  = mean_squared_error(y_test, y_pred)
    print("MSE :" , MSE)

    RMSE = np.sqrt(MSE)
    print("RMSE :" ,RMSE)

    MSE : 2265580.7289698995
    RMSE : 1505.1846162414429

[ ] r2 = r2_score(y_test, y_pred)
    print("R2 :" ,r2)

    R2 : 0.7634756515972615

[ ] pd.DataFrame(zip(y_test, y_pred), columns = ['actual', 'pred'])
```

	actual	pred
0	5495	5324.460835
1	5472	4849.366393

Fig 25 :- Linear Regression on Medium Enterprises data without 0

LASSO

```
[ ] from sklearn.model_selection import cross_val_score
    L1 = Lasso(alpha = 0.4, max_iter=10000,selection='cyclic', tol=0.0001,)

    L1.fit(X_train, y_train)

    Lasso
    Lasso(alpha=0.4, max_iter=10000)

[ ] y_pred_lasso = L1.predict(X_test)

[ ] L1.score(X_test, y_test)

    0.7634664345398297

[ ] cv_scores = cross_val_score(L1, X, y, cv=10)
    mean_cv_score = cv_scores.mean()

[ ] cv_scores

    array([0.73875078, 0.74886069, 0.76075568, 0.75799535, 0.75829597,
           0.76481931, 0.76724952, 0.75273712, 0.75812694, 0.78479044])

[ ] mean_cv_score

    0.7584381798257628

[ ] from sklearn.model_selection import GridSearchCV, cross_val_score
    # define the range of alpha values to test

from sklearn.model_selection import GridSearchCV, cross_val_score
# define the range of alpha values to test
parameters = {'alpha': [0.1, 0.2, 0.3, 0.4, 0.5]}

# perform grid search to find the best alpha value
lasso_cv = GridSearchCV(L1, parameters, cv=5)
lasso_cv.fit(X, y)

GridSearchCV
estimator: Lasso
Lasso

# extract the best alpha value and corresponding score
best_alpha_lasso = lasso_cv.best_params_['alpha']
best_score_lasso= lasso_cv.best_score_

best_alpha_lasso

0.2

best_score_lasso

0.7605710673189068

pd.DataFrame(zip(y_test, y_pred_lasso), columns = ['actual', 'pred'])
```

	actual	pred
0	5495	5326.552968

Fig 26:- Lasso Regression on Medium enterprises data without 0

Decision Tree

```
[ ] sales_mean=final1[dependent_variables].mean()

[ ] sales_mean

    5773.822502553556

[ ] sales_mean_new=new_df[dependent_variables].mean()

[ ] sales_mean_new

    6955.518543520071

[ ] decision_tree=DecisionTreeRegressor(max_depth=5)
    decision_tree.fit(X_train, y_train)
    y_pred_dt = decision_tree.predict(X_test)
    y_train_dt = decision_tree.predict(X_train)
    #print('dt_regressor R^2: ', r2(v_test,v_pred))
    MSE  = mean_squared_error(y_test, y_pred_dt)
    print("MSE :" , MSE)

    RMSE = np.sqrt(MSE)
    print("RMSE :" ,RMSE)

    RMPSE=RMSE/sales_mean_new
    print("RMPSE :",RMPSE)

    r2 = r2_score(y_test, y_pred_dt)
    print("R2 :" ,r2)

    MSE : 2006720.4154441394
    RMSE : 1416.5875954010537
```

Fig 27: Decision Tree on Medium Dataset without 0

LINEAR REGRESSION

```
[ ] # scaling the x values
    scaler=StandardScaler()

    U_train = scaler.fit_transform(U_train)
    U_test = scaler.transform(U_test)

[ ] # fitting the data into Linear Regression Model
    linear_regression = LinearRegression()
    linear_regression.fit(U_train, v_train)

    ▸ LinearRegression
    LinearRegression()

[ ] v_pred=linear_regression.predict(U_test)
    v_pred

    array([ 7096.31153835, 12600.9482571 ,  9195.6982571 , ...,
            6936.57716335,  5998.06310085,  4001.00739773])

[ ] linear_regression.score(U_train, v_train)

    0.8677971058979633

[ ] regression_Dataframe = pd.DataFrame(zip(v_test, v_pred), columns = ['actual', 'pred'])
    regression_Dataframe
```

	actual	pred
0	7285	7096.311538

Fig 28: Linear Regression on Whole Dataset of Medium Enterprises

```
decision_tree=DecisionTreeRegressor(max_depth=5)
decision_tree.fit(U_train, v_train)
v_pred_dt = decision_tree.predict(U_test)
v_train_dt = decision_tree.predict(U_train)
#print('dt_regressor R^2: ', r2(v_test,v_pred))
MSE  = mean_squared_error(v_test, v_pred_dt)
print("MSE :" , MSE)

RMSE = np.sqrt(MSE)
print("RMSE :" ,RMSE)

RMPSE=RMSE/sales_mean
print("RMPSE :",RMPSE)

r2 = r2_score(v_test, v_pred_dt)
print("R2 :" ,r2)
```

```
MSE : 1938824.9966190814
RMSE : 1392.4169621988528
RMPSE : 0.2411603338313633
R2 : 0.8687929764541089
```

```
decisiontree_Dataframe = pd.DataFrame(zip(v_test, v_pred_dt), columns = ['actual', 'pred'])
decisiontree_Dataframe
```

	actual	pred
0	7285	6405.437098
1	6221	10731.782531
2	8132	9096.412211
3	20916	11835.129880
4	5472	5476.684725

Fig 29:- Decision Tree on whole Dataset of Medium Enterprise

Random Forest

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Create a random forest regressor with n_estimators=500, max_depth=8, and n_jobs=2
random_forest = RandomForestRegressor(n_estimators=500, max_depth=8, n_jobs=2)

# Fit the random forest to the training data
random_forest.fit(U_train, v_train)

# Make predictions on the test data
v_pred_rf = random_forest.predict(U_test)

# Calculate the mean squared error (MSE) between the predicted and actual values
MSE = mean_squared_error(v_test, v_pred_rf)
print("MSE:", MSE)

# Calculate the root mean squared error (RMSE)
RMSE = np.sqrt(MSE)
print("RMSE:", RMSE)

# Calculate the root mean squared percentage error (RMPSE)
sales_mean = np.mean(v_test)
RMPSE = RMSE / sales_mean
print("RMPSE:", RMPSE)

# Calculate the coefficient of determination (R2 score)
r2 = r2_score(v_test, v_pred_rf)
print("R2:", r2)
```

```
MSE: 1116360.4077130198
RMSE: 1056.5795794510793
RMPSE: 0.18284263046850002
```

Fig 30 : Random Forest Regression on Whole dataset

XGBBoost

```
import xgboost as xgb

xgboost = xgb.XGBRegressor(n_estimators=500, max_depth=8, n_jobs=2)
xgboost.fit(U_train, v_train)
v_pred_xgb = xgboost.predict(U_test)

MSE = mean_squared_error(v_test, v_pred_xgb)
print("MSE :", MSE)

RMSE = np.sqrt(MSE)
print("RMSE :", RMSE)

RMPSE = RMSE/sales_mean
print("RMPSE :", RMPSE)

r2 = r2_score(v_test, v_pred_xgb)
print("R2 :", r2)
```

```
MSE : 177443.19553202452
RMSE : 421.24006876367366
RMPSE : 0.07289620557639033
R2 : 0.9879918024706577
```

Fig 31 : XGBoost regression on Medium Enterprise Dataset

Linear Regression

```
def score(model, X=X, y=y):
    print("Average R2 Score :", np.average(cross_val_score(model, X, y, cv=10)))
    print("Average Root Mean Square Error :", np.average(cross_val_score(model, X, y, cv=10, scoring='neg_root_mean_squared_error')))
```

```
from yellowbrick.regressor import prediction_error
```

```
from yellowbrick.regressor import ResidualsPlot

def residuals_plot(model, X_train, y_train, X_test, y_test):
    visualizer = ResidualsPlot(model)
    visualizer.fit(X_train, y_train)
    visualizer.score(X_test, y_test)
    visualizer.show()
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
# Linear Regression

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score

LR = LinearRegression(positive=True)
LR.fit(X,y)

score(LR)
```

```
Average R2 Score : 0.4825351933951542
Average Root Mean Square Error : -1225.8561623734372
```

Fig 32: Linear Regression on Small Enterprise Dataset

Lasso Regression

```python
from sklearn.linear_model import Lasso
from yellowbrick.regressor import ManualAlphaSelection

alphas = np.logspace(0, 0.35, 50)

visualizer = ManualAlphaSelection(
    Lasso(positive=True),
    alphas=alphas,
    cv=12,
    scoring="r2"
)

visualizer.fit(X, y)
visualizer.show()
```

```python
from sklearn.metrics import mean_squared_error

ls = Lasso(alpha=1.58, positive=True)
ls.fit(X,y)

score(ls)
```

```
Average R2 Score : 0.4825553592709074
Average Root Mean Square Error : -1225.848966067305
```

Fig 33: Lasso Regression on Small Enterprise Dataset

```python
from yellowbrick.model_selection import ValidationCurve
from sklearn.ensemble import RandomForestRegressor

viz = ValidationCurve(
    RandomForestRegressor(), param_name="max_depth",
    param_range=np.arange(1, 20), cv=5, scoring="r2"
)

viz.fit(X, y)
viz.show()
```

```python
viz = ValidationCurve(
    RandomForestRegressor(), param_name="max_depth",
    param_range=np.arange(1, 8), cv=5, scoring="r2"
)

viz.fit(X, y)
viz.show()
```

```python
rfr = RandomForestRegressor(max_depth=5, random_state=5)
rfr.fit(X,y)

score(rfr)
```

```
Average R2 Score : 0.595680819809614
Average Root Mean Square Error : -1082.8528603821146
```

Fig 34 : Random Forest Regression on Small Enterprise Dataset

References

Liu, X. &. (2017). Food sales prediction with meteorological data — A case study of a Japanese chain supermarket. Data Mining and Big Data: Second International Conference, DMBD 2017, Fukuoka, Japan, 93-104.

Zhuang Q, Z. X. (2019). A Neural Network Model for China B2C E-Commerce Sales Forecast Based on Promotional Factors and Historical Data. International Conference on Economic Management and Model Engineering (ICEMME). IEEE, 307-312.

Huber, J. &. (2020). Daily retail demand forecasting using machine learning with emphasis on calendric special days. International Journal of Forecasting, 1420-1438.

Florian Haselbeck, J. K. (2022). Machine Learning Outperforms Classical Forecasting on Horticultural Sales Prediction. Machine Learning with Applications.