# A PROJECT REPORT

# ON

# BIGMART SALES PREDICTIVE MODEL

Submitted in partial fulfillment for the requirement of the award of

Training

in

Data Analytics, Machine Learning and AI using Python

Diginique Techlabs
Ideal for Geeks

*Submitted By*

**Meera Srilekha.B**

(B Tech CSE - II year)

**(**Karunya Institute of Technology and Sciences, Coimbatore,Tamil Nadu**)**

# ACKNOWLEDGEMENT

My sincere gratitude and thanks towards my training and project guide Ashish Saini.

It was only with his backing and support that I could complete the report. He provided me with all sorts of help and corrected me if ever seemed to make mistakes. I acknowledge my sincere gratitude to the lecturer for their valuable guidance and helping attitude even in their very busy schedule. And at last but not the least, I acknowledge my dearest parents for being such a nice source of encouragement and moral support that helped me tremendously in this aspect. I also declare to the best of my knowledge and belief that the Project Work has not been submitted anywhere else.


## INTRODUCTION

BigMart is a large retail chain that operates stores in India. The company collects a lot of data about its customers and products, including sales data, customer demographics, and product attributes. This data can be used to build predictive models that can help the company make better decisions about product pricing, promotions, and inventory levels.

BigMart sales predictive model is a model which aims to predict the stock rate of each product in the store. By forecasting the stock rate , bigmart can improve its supply chain management and inventory management. This project deals with providing information about the product's stock rate, fat content in it, MRP and outlet size . The model is trained on a dataset of over 200,000 sales transactions, and it has been shown to be very accurate in its predictions. Stores are benefited using these predictive models. This predictive model mainly uses python as its base programming language. This model uses flask for web deployment. Python has various in-built libraries which can be used to make various analyses for bigmart sales.

The predictive model deals with a set of procedures in it, such as data preprocessing, data cleansing, EDA, label encoding, splitting up train and test data set, standardization, model building and hyper parameter tuning and lastly saving the model. Each process in this predictive model deals with cleansing , analyzing, and standardizing the dataset.

## PROBLEM STATEMENT

The Bigmart sales predictive model aims at accurately predicting the future sales of products.

## TECHNOLOGY AND CONCEPTS USED

1. Data preparation
    - Data manipulation
    - Data blending
    - Missing values handling
    - Feature generation
    - Feature selection
    - Data cleansing
2. Model training
    - Using ensembles
3. Model optimization
4. Model evaluation
5. Web deployment with flask

Python libraries used in this predictive model,

Libraries used/can be used for analysis,

- numpy
- pandas
- matplotlib
- seaborn

Libraries used/can be used for EDA,

- Dtale
- SweetViz
- Klib
- pandas_profiling

## DESCRIPTION OF THE DATASET

The dataset comprises several features that are essential for developing an accurate predictive model. Each row represents a unique sales transaction at different Big Mart outlets, and the dataset contains the following key attributes:

Item Identifier: A unique identifier for each product sold at Big Mart.
Item Weight: The weight of the product.
Item Visibility: A measure of the visibility of the product within the store.
Item Type: The category or type of product.
Item MRP (Maximum Retail Price): The maximum price at which the product is sold to customers.

Outlet Identifier: A unique identifier for each Big Mart outlet.
Outlet Establishment Year: The year in which the outlet was established.
Outlet Size: The size category of the store (small, medium, or large).
Outlet Location Type: The type of location where the store is situated (urban or rural).
Outlet Type: The type of outlet (grocery store or supermarket).
Item Outlet Sales: The sales of the product (target variable).

## DRAWBACKS

- Change of customer behavior will lead to less accurate predictions as the model may not adapt quickly with respect to the customer behavior.
- Seasonality and trends may affect this predictive model.
- Model overfitting can produce some noise and random variations in the predictions as it is trained on either complex dataset or on limited dataset.

## EXPERIMENTAL SETUP

- Code
- analysis
- flask deployment

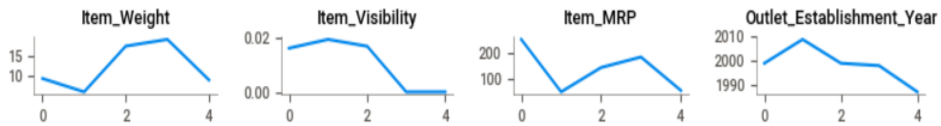
## CODE

## PLOTS OF THE DATASET

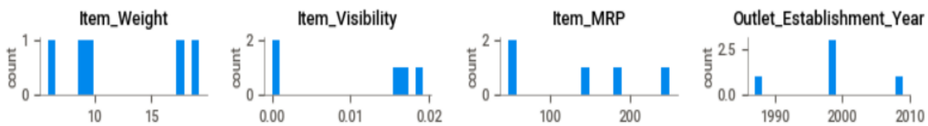
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
df_train = pd.read_csv('train.csv')
df_test=pd.read_csv('test.csv')
df_train.head()
```

| | Item_Identifier | Item_Weight | Item_Fat_Content | Item_Visibility | Item_Type | Item_MRP | Outlet_Identifier | Outlet_Establishment_Year | Outlet_Size |
|---|---|---|---|---|---|---|---|---|---|
| 0 | FDA15 | 9.30 | Low Fat | 0.016047 | Dairy | 249.8092 | OUT049 | 1999 | Medium |
| 1 | DRC01 | 5.92 | Regular | 0.019278 | Soft Drinks | 48.2692 | OUT018 | 2009 | Medium |
| 2 | FDN15 | 17.50 | Low Fat | 0.016760 | Meat | 141.6180 | OUT049 | 1999 | Medium |
| 3 | FDX07 | 19.20 | Regular | 0.000000 | Fruits and Vegetables | 182.0950 | OUT010 | 1998 | NaN |
| 4 | NCD19 | 8.93 | Low Fat | 0.000000 | Household | 53.8614 | OUT013 | 1987 | High |

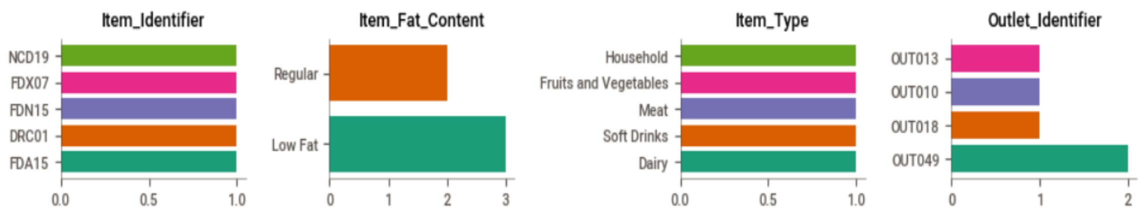
## Values



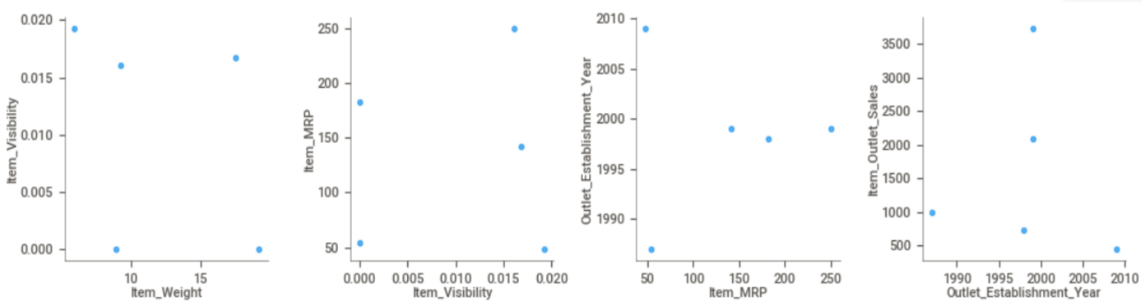

## Distributions



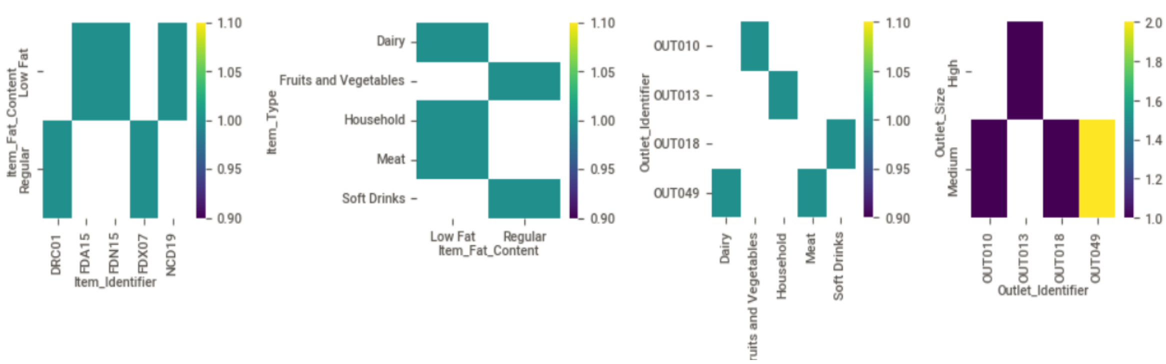

## Categorical distributions

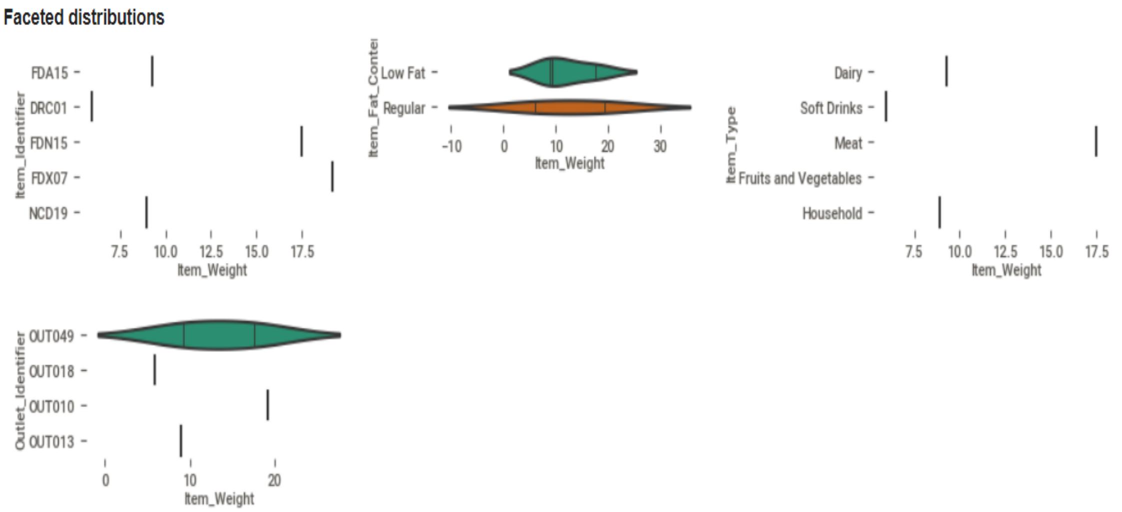



## 2-d distributions




## 2-d categorical distributions






```
df_train.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                  1463
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
Outlet_Location_Type            0
Outlet_Type                     0
Item_Outlet_Sales               0
dtype: int64
```

```
[5]  df_test.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                   976
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  1606
Outlet_Location_Type            0
Outlet_Type                     0
dtype: int64
```

[7]
```
df_train.describe()
```

| | Item_Weight | Item_Visibility | Item_MRP | Outlet_Establishment_Year | Item_Outlet_Sales |
|---|---|---|---|---|---|
| **count** | 7060.000000 | 8523.000000 | 8523.000000 | 8523.000000 | 8523.000000 |
| **mean** | 12.857645 | 0.066132 | 140.992782 | 1997.831867 | 2181.288914 |
| **std** | 4.643456 | 0.051598 | 62.275067 | 8.371760 | 1706.499616 |
| **min** | 4.555000 | 0.000000 | 31.290000 | 1985.000000 | 33.290000 |
| **25%** | 8.773750 | 0.026989 | 93.826500 | 1987.000000 | 834.247400 |
| **50%** | 12.600000 | 0.053931 | 143.012800 | 1999.000000 | 1794.331000 |
| **75%** | 16.850000 | 0.094585 | 185.643700 | 2004.000000 | 3101.296400 |
| **max** | 21.350000 | 0.328391 | 266.888400 | 2009.000000 | 13086.964800 |

[8]
```
df_train['Item_Weight'].fillna(df_train['Item_Weight'].mean(),inplace=True)
df_test['Item_Weight'].fillna(df_test['Item_Weight'].mean(),inplace=True)
```

```
df_train.isnull().sum()
```

```
Item_Identifier                 0
Item_Weight                     0
Item_Fat_Content                0
Item_Visibility                 0
Item_Type                       0
Item_MRP                        0
Outlet_Identifier               0
Outlet_Establishment_Year       0
Outlet_Size                  2410
```

[10]
```
df_train['Outlet_Size'].value_counts()
```

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

```
df_train['Outlet_Size'].fillna(df_train['Outlet_Size'].mode()[0],inplace=True)
df_test['Outlet_Size'].fillna(df_test['Outlet_Size'].mode()[0],inplace=True)
df_train.isnull().sum()
```

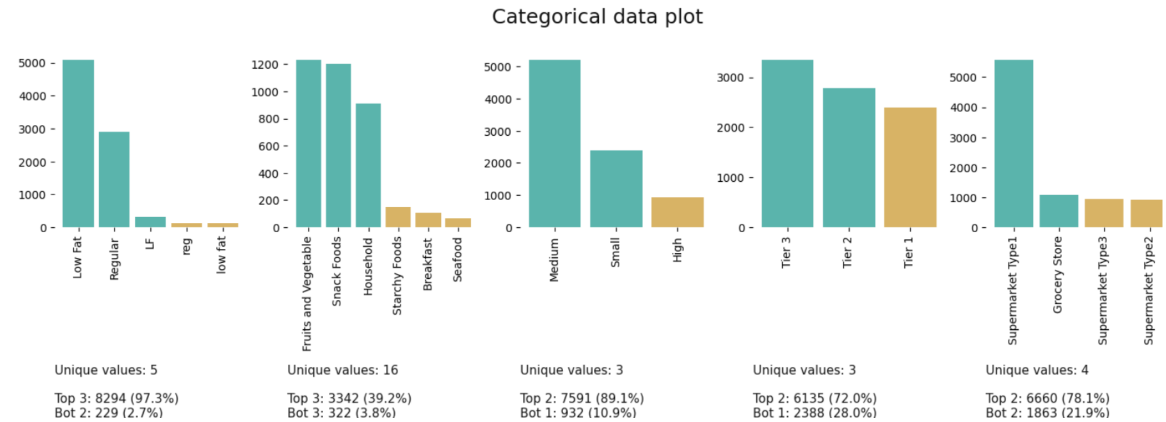
```
Item_Identifier              0
Item_Weight                  0
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  0
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

[12]
```
df_train.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
df_test.drop(['Item_Identifier','Outlet_Identifier'],axis=1,inplace=True)
```

# EDA WITH KLIB

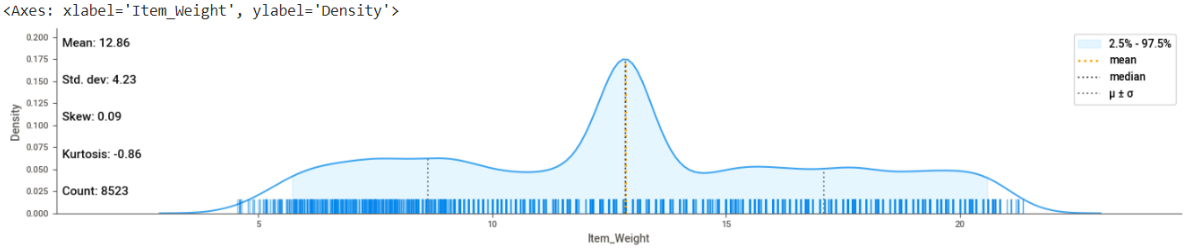
```
[14] import klib
     klib.cat_plot(df_train)
```

```
GridSpec(6, 5)
```




```
[27] klib.dist_plot(df_train)
```

```
<Axes: xlabel='Item_Weight', ylabel='Density'>
```



# EDA WITH PANDAS_PROFILING

```
[24] from pandas_profiling import ProfileReport
     profile = ProfileReport(df_train,title="pandas profiling report")
     print(profile)
```

```
2023-08-02 09:31:38,188 - INFO     - Pandas backend loaded 1.5.3
2023-08-02 09:31:38,199 - INFO     - Numpy backend loaded 1.22.4
2023-08-02 09:31:38,201 - INFO     - Pyspark backend NOT loaded
2023-08-02 09:31:38,203 - INFO     - Python backend loaded
```

```
[25] profile
```

Summarize dataset: 100% 44/44 [00:10<00:00, 2.34it/s, Completed]

Generate report structure: 100% 1/1 [00:07<00:00, 7.45s/it]

Render HTML: 100% 1/1 [00:01<00:00, 1.38s/it]

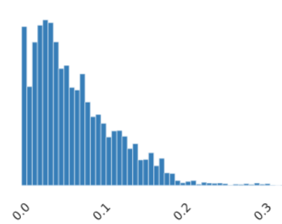
[25]

pandas profiling report — Overview | Variables | Interactions | Correlations | Missing values | Samp

Item_Visibility

Real number (ℝ)

| | | | |
|---|---|---|---|
| Distinct | 7880 | Minimum | 0 |
| Distinct (%) | 92.5% | Maximum | 0.32839095 |
| Missing | 0 | Zeros | 526 |
| Missing (%) | 0.0% | Zeros (%) | 6.2% |
| Infinite | 0 | Negative | 0 |
| Infinite (%) | 0.0% | Negative (%) | 0.0% |
| Mean | 0.066132029 | Memory size | 66.7 KiB |




More details

[25]

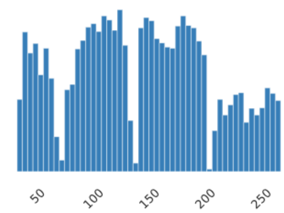
pandas profiling report | Overview | Variables | Interactions | Correlations | Missing values | Sampl

## Item_MRP

Real number (ℝ)

| | |
|---|---|
| **Distinct** | 5938 |
| **Distinct (%)** | 69.7% |
| **Missing** | 0 |
| **Missing (%)** | 0.0% |
| **Infinite** | 0 |
| **Infinite (%)** | 0.0% |
| **Mean** | 140.99278 |

| | |
|---|---|
| **Minimum** | 31.29 |
| **Maximum** | 266.8884 |
| **Zeros** | 0 |
| **Zeros (%)** | 0.0% |
| **Negative** | 0 |
| **Negative (%)** | 0.0% |
| **Memory size** | 66.7 KiB |



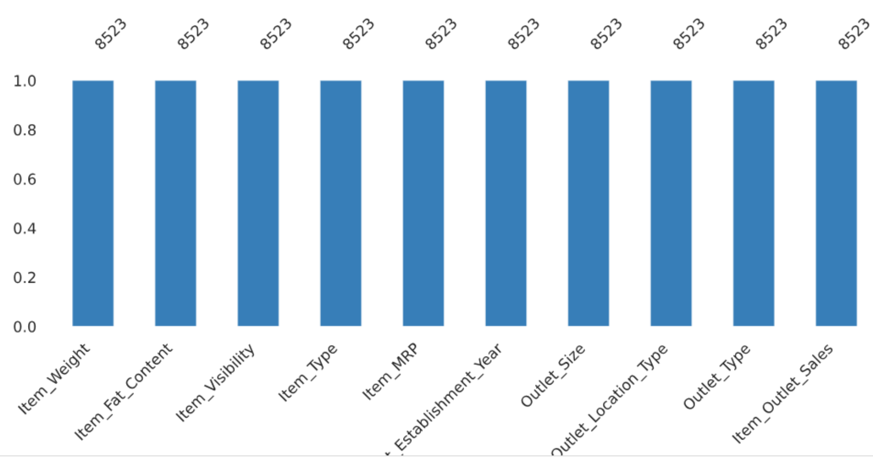

More details

[25]

pandas profiling report | Overview | Variables | Interactions | Correlations | Missing values | Samp

# Missing values

Count | Matrix



[25]

# Correlations

Auto

Heatmap Table

| | Item_Weight | Item_Visibility | Item_MRP | Outlet_Establishment_Year | Item_Outlet_Sales |
|---|---|---|---|---|---|
| **Item_Weight** | 1.000 | -0.009 | 0.026 | -0.023 | 0.013 |
| **Item_Visibility** | -0.009 | 1.000 | 0.006 | -0.055 | -0.115 |
| **Item_MRP** | 0.026 | 0.006 | 1.000 | 0.004 | 0.563 |
| **Outlet_Establishment_Year** | -0.023 | -0.055 | 0.004 | 1.000 | 0.043 |
| **Item_Outlet_Sales** | 0.013 | -0.115 | 0.563 | 0.043 | 1.000 |
| **Item_Fat_Content** | 0.049 | 0.035 | 0.043 | 0.011 | 0.000 |
| **Item_Type** | 0.080 | 0.063 | 0.087 | 0.000 | 0.003 |
| **Outlet_Size** | 0.085 | 0.059 | 0.000 | 0.641 | 0.075 |

[26]
```
plt.figure(figsize=(10,5))
sns.heatmap(df_train.corr(),annot = True)
plt.show()
```

```
<ipython-input-26-9eedae251a34>:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Se
```

# DATA CLEANSING USING KLIB

```
#data cleansing using klib
klib.data_cleaning(df_train)
```

```
Shape of cleaned data: (8523, 10) - Remaining NAs: 0


Dropped rows: 0
     of which 0 duplicates. (Rows (first 150 shown): [])

Dropped columns: 0
     of which 0 single valued.     Columns: []
Dropped missing values: 0
Reduced memory by at least: 0.46 MB (-70.77%)
```

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type |
|---|---|---|---|---|---|---|---|---|
| 0 | 9.300000 | Low Fat | 0.016047 | Dairy | 249.809204 | 1999 | Medium | Tier 1 |
| 1 | 5.920000 | Regular | 0.019278 | Soft Drinks | 48.269199 | 2009 | Medium | Tier 3 |
| 2 | 17.500000 | Low Fat | 0.016760 | Meat | 141.617996 | 1999 | Medium | Tier 1 |
| 3 | 19.200001 | Regular | 0.000000 | Fruits and Vegetables | 182.095001 | 1998 | Medium | Tier 3 |
| 4 | 8.930000 | Low Fat | 0.000000 | Household | 53.861401 | 1987 | High | Tier 3 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 8518 | 6.865000 | Low Fat | 0.056783 | Snack Foods | 214.521805 | 1987 | High | Tier 3 |

```
klib.convert_datatypes(df_train)
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8523 entries, 0 to 8522
Data columns (total 10 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   item_weight                8523 non-null   float64
 1   item_fat_content           8523 non-null   object
 2   item_visibility            8523 non-null   float64
 3   item_type                  8523 non-null   object
 4   item_mrp                   8523 non-null   float64
 5   outlet_establishment_year  8523 non-null   int64
 6   outlet_size                8523 non-null   object
 7   outlet_location_type       8523 non-null   object
 8   outlet_type                8523 non-null   object
 9   item_outlet_sales          8523 non-null   float64
dtypes: float64(4), int64(1), object(5)
memory usage: 666.0+ KB
```

# PREPROCESSING THE DATA

# LABEL ENCODING

```
#preprocessing task before model building label encoding

from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df_train.head()
df_train['item_fat_content']= le.fit_transform(df_train['item_type'])
df_train['item_type']= le.fit_transform(df_train['item_fat_content'])
df_train['outlet_size']= le.fit_transform(df_train['outlet_size'])
df_train['outlet_location_type']= le.fit_transform(df_train['outlet_location_type'])
df_train['outlet_type']= le.fit_transform(df_train['outlet_type'])
df_train.head()
```

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type |
|---|---|---|---|---|---|---|---|---|
| **0** | 9.30 | 4 | 0.016047 | 4 | 249.8092 | 1999 | 1 | 0 |
| **1** | 5.92 | 14 | 0.019278 | 14 | 48.2692 | 2009 | 1 | 2 |
| **2** | 17.50 | 10 | 0.016760 | 10 | 141.6180 | 1999 | 1 | 0 |
| **3** | 19.20 | 6 | 0.000000 | 6 | 182.0950 | 1998 | 1 | 2 |
| **4** | 8.93 | 9 | 0.000000 | 9 | 53.8614 | 1987 | 0 | 2 |

# SPLITTING THE DATA INTO TRAIN AND TEST

```
[37] #spliting data into train and test
X= df_train.drop('item_outlet_sales',axis=1)
Y=df_train['item_outlet_sales']
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,random_state=101,test_size=0.2)
```

```
X_train
```

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | ou |
|---|---|---|---|---|---|---|---|---|
| **3684** | 19.250 | 4 | 0.101689 | 4 | 54.6956 | 1987 | 0 | |
| **1935** | 7.630 | 14 | 0.061410 | 14 | 94.6436 | 2007 | 1 | |
| **5142** | 19.350 | 10 | 0.065891 | 10 | 167.0816 | 2007 | 1 | |
| **4978** | 6.380 | 4 | 0.031898 | 4 | 177.4344 | 1997 | 2 | |
| **2299** | 16.700 | 4 | 0.022110 | 4 | 110.8886 | 2002 | 1 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **599** | 5.000 | 14 | 0.044005 | 14 | 188.8530 | 1997 | 2 | |
| **5695** | 14.650 | 7 | 0.170664 | 7 | 56.4614 | 2002 | 1 | |
| **8006** | 12.500 | 8 | 0.018849 | 8 | 96.7384 | 1997 | 2 | |
| **1361** | 9.695 | 0 | 0.129009 | 0 | 226.9404 | 2007 | 1 | |
| **1547** | 15.700 | 9 | 0.161317 | 9 | 57.5562 | 2009 | 1 | |

## STANDARDIZATION

```
[46] from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()

     X_train_std=scaler.fit_transform(X_train)
     X_test_std=scaler.transform(X_test)
```

```
X_train_std
```

```
array([[ 1.52290029, -0.75847359,  0.68469729, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.23985603,  1.60759199, -0.09514748, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 1.54667616,  0.66116576, -0.00838589, ..., -0.28872895,
        -0.13870429, -0.25964107],
       ...,
       [-0.08197107,  0.18795264, -0.9191623 , ...,  1.37953713,
        -1.36527477, -0.25964107],
       [-0.74888428, -1.70489982,  1.21363058, ..., -0.28872895,
        -0.13870429, -0.25964107],
       [ 0.67885683,  0.4245592 ,  1.83915356, ..., -0.28872895,
         1.08786619,  0.98524841]])
```

```
[48] X_test_std
```

```
array([[-0.43860915,  0.18795264, -0.21609255, ..., -0.28872895,
         1.08786619,  0.98524841],
       [ 1.22570189,  1.37098543, -0.52943461, ..., -1.95699503,
         1.08786619, -0.25964107],
       [-1.21845775, -1.46829326,  0.16277342, ...,  1.37953713,
        -1.36527477, -0.25964107],
```

# MODEL BUILDING

[52]
```
#model building
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
```

```
lr.fit(X_train_std,Y_train)
```

```
▾ LinearRegression
LinearRegression()
```

[54]
```
lr.predict(X_test_std)
```

```
array([2058.70486727, 2138.97070243, 1191.34604755, ..., 1286.92741319,
       2374.1049276 , 2331.06383235])
```

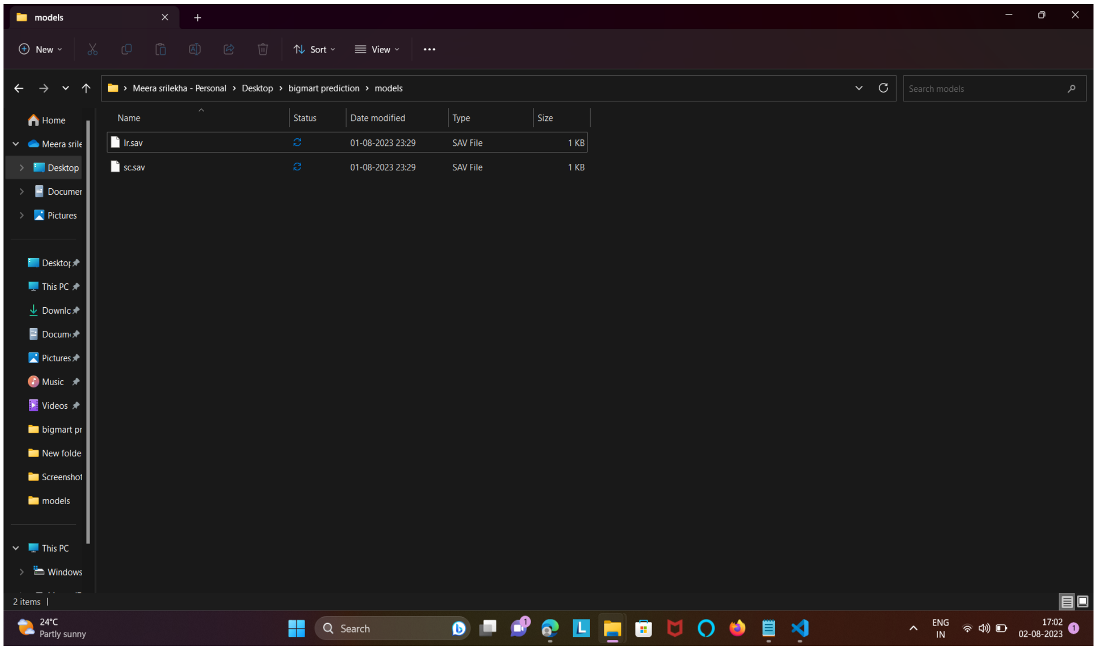
[55]
```
X_test.head()
```

| | item_weight | item_fat_content | item_visibility | item_type | item_mrp | outlet_establishment_year | outlet_size | outlet_location_type |
|---|---|---|---|---|---|---|---|---|
| **8179** | 11.00 | 8 | 0.055163 | 8 | 100.3358 | 2009 | 1 | 2 |
| **8355** | 18.00 | 13 | 0.038979 | 13 | 148.6418 | 1987 | 0 | 2 |
| **3411** | 7.72 | 1 | 0.074731 | 1 | 77.5986 | 1997 | 2 | 0 |
| **7089** | 20.70 | 6 | 0.049035 | 6 | 39.9506 | 2007 | 1 | 1 |
| **6954** | 7.55 | 3 | 0.027225 | 3 | 152.9340 | 2002 | 1 | 1 |

# HYPER PARAMETER TUNING

[67]
```
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

# define models and parameters
model = RandomForestRegressor()
n_estimators = [10, 100, 1000]
max_depth=range(1,31)
min_samples_leaf=np.linspace(0.1, 1.0)
max_features=["auto", "sqrt", "log2"]
min_samples_split=np.linspace(0.1, 1.0, 10)

# define grid search
grid = dict(n_estimators=n_estimators)

#cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3, random_state=101)

grid_search_forest = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                           scoring='r2',error_score=0,verbose=2,cv=2)

grid_search_forest.fit(X_train_std, Y_train)

# summarize results
print(f"Best: {grid_search_forest.best_score_:.3f} using {grid_search_forest.best_params_}"
means = grid_search_forest.cv_results_['mean_test_score']
stds = grid_search_forest.cv_results_['std_test_score']
params = grid_search_forest.cv_results_['params']

for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")
```

## SAVE THE MODEL USING JOBLIB




Two SAV files are saved in this folder named bigmart prediction.

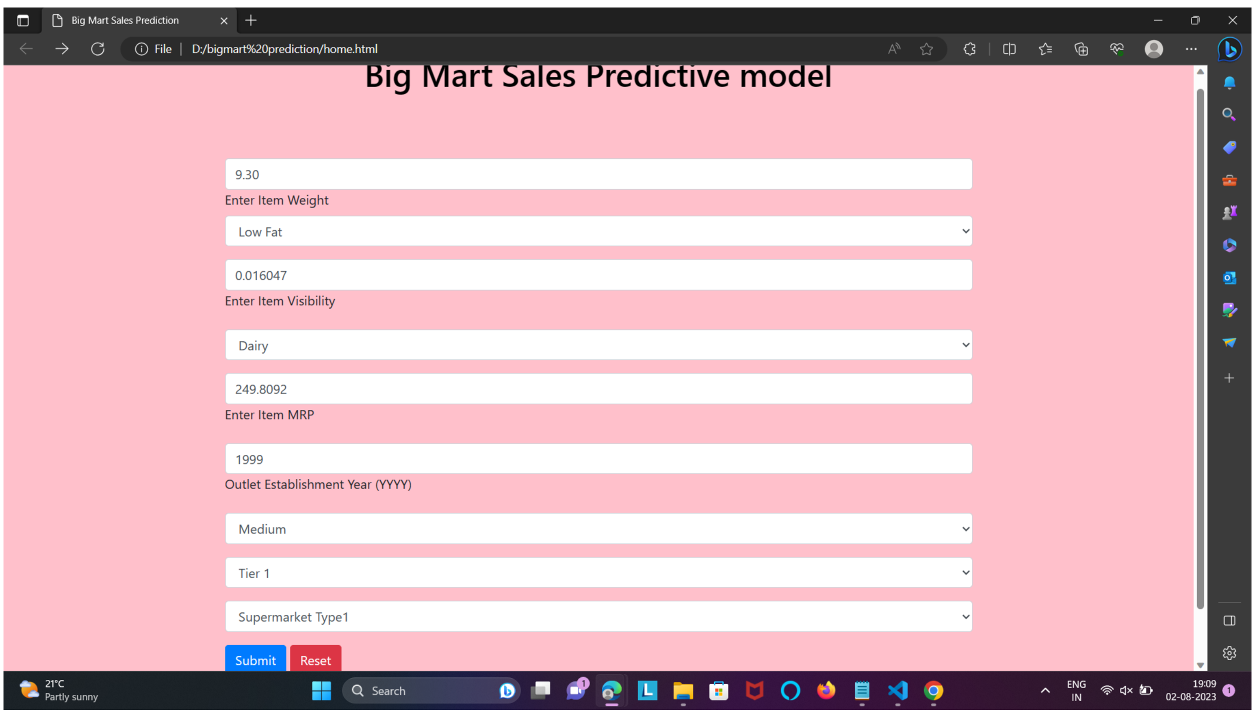
## GOOGLE COLAB LINK FOR ACCESSING THE CODE

**Bigmart sales predictive model.ipynb**

## FLASK DEPLOYMENT

```
from flask import Flask, jsonify, render_template, request
import joblib
import os
import numpy as np

app = Flask(__name__)



@app.route("/")
def index():
    return render_template("home.html")

@app.route('/predict',methods=['POST','GET'])
def result():

    item_weight= float(request.form['item_weight'])
    item_fat_content=float(request.form['item_fat_content'])
    item_visibility= float(request.form['item_visibility'])
    item_type= float(request.form['item_type'])
    item_mrp = float(request.form['item_mrp'])
    outlet_establishment_year= float(request.form['outlet_establishment_year'])
    outlet_size= float(request.form['outlet_size'])
    outlet_location_type= float(request.form['outlet_location_type'])
    outlet_type= float(request.form['outlet_type'])

    X= np.array([[ item_weight,item_fat_content,item_visibility,item_type,item_mrp,
              outlet_establishment_year,outlet_size,outlet_location_type,outlet_type ]])

    scaler_path= r'C:\Users\B.Meera Srilekha\OneDrive\Desktop\bigmart prediction\models\sc.sav'

    sc=joblib.load(scaler_path)
```

```
    X_std= sc.transform(X)

    model_path=r'C:\Users\B.Meera Srilekha\OneDrive\Desktop\bigmart prediction\models\lr.sav'

    model= joblib.load(model_path)

    Y_pred=model.predict(X_std)

    return jsonify({'Prediction': float(Y_pred)})

if __name__ == "__main__":
    app.run(debug=True, port=9457)
```



## DIFFICULTIES

- There were two many libraries to do the analysis part so choosing the best amongst them was challenging.
- Proper procedure must be followed in order to make the predictive model if not the process cannot be developed further.
- The data should be stored in some SAV file, but the content was not saved in the file. That is the challenging portion where everything from  first needs to be checked.

## CONCLUSIONS

The Big Mart Sales Predictive Model project has been a significant endeavor to develop an efficient and accurate forecasting tool for Big Mart stores. Through the rigorous analysis of historical sales transactions, product attributes, and store details, I have successfully created a predictive model that can provide valuable insights and support data-driven decision-making.

The predictive model demonstrates strong performance in forecasting sales, with low Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) values on the testing dataset, indicating its reliability and accuracy. This is a great project where it implies to follow the procedures of the data analysis part and even get to know more about the predictive model's process.

This predictive model's functionalities can be extended further by using more advanced technologies for productive management of bigmart sales.

## BIBLIOGRAPHY


**Bigmart Dataset Sales Prediction. This post is about my approach on… | by Vishal Borana | Analytics Vidhya | Medium**

**Big_Data_Sales_Prediction/Big_Mart_Sales_Prediction.ipynb at main · sowmyavarshinipathala/Big_Data_Sales_Prediction (github.com)**