**Name:Vikas Mane**

**Roll No:B1921152**

**Class: BE-B**

Assignment 3

**Title: Big-Mart Sales Analysis:** For data comprising of transaction records of a sales store.

**Aim:** Predict the sales of a store. The data has 8523 rows of 12 variables.

**System Requirements:** Python, Data Visualization Libraries, Jupyter Notebook.

# Theory:

* In today’s modern world, huge shopping centers such as big malls and marts are recording data related to sales of items or products with their various dependent or independent factors as an important step to be helpful in prediction of future demands and inventory management. The dataset built with various dependent and independent variables is a composite form of item attributes, data gathered by means of customer, and also data related to inventory management in a data warehouse. The data is thereafter refined in order to get accurate predictions and gather new as well as interesting results that shed a new light on our knowledge with respect to the task’s data. This can then further be used for forecasting future sales by means of employing machine learning algorithms such as the random forests and simple or multiple linear regression model.
* BigMart’s data scientists collected sales data of their 10 stores situated at different locations with each store having 1559 different products as per 2013 data collection. Using all the observations it is inferred what role certain properties of an item play and how they affect their sales.
* On implementation, the prediction results show the correlation among different attributes considered and how a particular location of medium size recorded the highest sales, suggesting that other shopping locations should follow similar patterns for improved sales. Multiple instances parameters and various factors can be used to make this sales prediction more innovative and successful. Accuracy, which plays a key role in prediction-based systems, can be significantly increased as the number of parameters used are increased. Also, a look into how the sub-models work can lead to increase in productivity of system

**Objective:**

To analyze the sales of a store

**Input:**

Structured Dataset : Big Mart Sales

**Output:**

1. Histograms / Bar Graphs for Analysis
2. Mean Squared Error using Bayesian Linear Regression

**Result / Conclusion:**

Hence, we have completed the analysis of Big Mart Store.

**Program:**

import numpy as np import pandas as pd

from matplotlib import pyplot as plt from numpy import linalg

trainDf = pd.read\_csv("train.csv") testDf = pd.read\_csv("test.csv") trainingSetIndex = len(trainDf)

trainDf.head() trainDf.dtypes print(trainDf.isnull().sum())

combination = trainDf.append(testDf)

combination = combination.drop(["Item\_Identifier", "Outlet\_Identifier"], axis=1)

# replacing the null in the ItemWeight

combination = combination.fillna(combination.median())

# replacing nominal values

combination["Item\_Fat\_Content"] = combination["Item\_Fat\_Content"].replace({"LF": 0, "reg": 1})

combination["Item\_Fat\_Content"] = combination["Item\_Fat\_Content"].replace({"Low Fat": 0, "Regular": 1})

combination["Item\_Fat\_Content"] = combination["Item\_Fat\_Content"].replace({"low fat": 0, "Regular": 1})

perishable = ["Breads", "Breakfast", "Dairy", "Fruits and Vegetables", "Meat", "Seafood"]

non\_perishable = ["Baking Goods", "Canned", "Frozen Foods", "Hard Drinks", "Health and Hygiene", "Household", "Soft Drinks", "Snack Foods", "Starchy Foods", "Others"]

combination["Item\_Type"] = combination["Item\_Type"].replace(to\_replace=perishable, value="perishable")

combination["Item\_Type"] = combination["Item\_Type"].replace(to\_replace=non\_perishable, value="non\_perishable") combination["Item\_Type"] = combination["Item\_Type"].replace({"perishable": 0, "non\_perishable": 1})

combination["Outlet\_Size"] = combination["Outlet\_Size"].replace({"Small": 0, "High": 1, "Medium": 2, np.nan: 3})

combination["Outlet\_Location\_Type"] = combination["Outlet\_Location\_Type"].replace({"Tier 3": 0, "Tier 2": 1, "Tier 1":

2})

combination["Outlet\_Type"] = combination["Outlet\_Type"].replace({"Grocery Store": 0,"Supermarket Type1": 1, "Supermarket Type2": 2, "Supermarket Type3": 3})

# splitting again the cleaned data sets trainDfClean = combination[:trainingSetIndex] testDfClean = combination[trainingSetIndex:]

trainDfClean.head() testDfClean.head()

trainDfClean.hist(bins=50, figsize=(20,15)); plt.show();

X\_train = trainDfClean.drop(["Item\_Outlet\_Sales"], axis=1).values

y\_train = trainDfClean["Item\_Outlet\_Sales"].values

X\_test = testDfClean.drop(["Item\_Outlet\_Sales"], axis=1).values y\_test = testDfClean["Item\_Outlet\_Sales"].values

from sklearn.linear\_model import LinearRegression from sklearn.metrics import mean\_squared\_error from sklearn.metrics import mean\_absolute\_error lreg = LinearRegression(); lreg.fit(X\_train,y\_train);

y\_pred = lreg.predict(X\_test); mean\_absolute\_error(y\_test, y\_pred) mean\_squared\_error(y\_test, y\_pred)

df = pd.DataFrame(list(zip(y\_test, y\_pred)), columns =['Actual', 'Predicted']) df.head(100)

def add\_intercept(X):

X\_new = np.ones((X.shape[0], X.shape[1] + 1), dtype=X.dtype) X\_new[:, 1:] = X[:, :]

return X\_new

class BayesianLinearRegression: """

Linear regression model: y = z beta[1] + beta[0] beta ~ N(0,Lambda)

Lambda = I \* lambda

P(y|x,beta) ~ N(y|x.dot(beta),sigma\*\*2) """

def init (self, lamb=20., beta\_mu=0, sigma=5, fit\_intercept=True): """

lamb: variance of the prior for each of the feature dimensions. beta\_mu: the mean of the prior

sigma: variance of the prediction error. """

if not np.isscalar(lamb):

self.inv\_lamb = 1. / np.asarray(lamb) else:

self.inv\_lamb = 1. / float(lamb) if not np.isscalar(beta\_mu):

self.beta\_mu = np.asarray(beta\_mu) else:

self.beta\_mu = float(beta\_mu)

self.sigma = sigma self.fit\_intercept = fit\_intercept self.beta = None

def fit\_ml(self, X, y): """

Fit a Maximum Likelihood estimate. (not Bayesian)

X: features, n\_samples by n\_features nd-array y: target values, n\_samples array

"""

if self.fit\_intercept: X = add\_intercept(X)

self.beta = linalg.inv(X.T.dot(X)).dot(X.T.dot(y))

def fit\_map(self, X, y): """

Fit a MAP estimate

X: features, n\_samples by n\_features nd-array y: target values, n\_samples array

"""

if self.fit\_intercept: X = add\_intercept(X)

# data setup f\_dim = X.shape[1]

if np.isscalar(self.inv\_lamb):

inv\_lamb = np.diagflat(np.repeat(self.inv\_lamb, f\_dim)) else:

inv\_lamb = np.diagflat(self.inv\_lamb) if np.isscalar(self.beta\_mu):

beta\_mu = np.repeat(self.beta\_mu, f\_dim) else:

beta\_mu = self.beta\_mu sigma = self.sigma

# let the actual calculation begin l = sigma \*\* 2 \* inv\_lamb

s = linalg.inv(X.T.dot(X) + l)

# adding in the mean of the prior

b0 = sigma \*\* 2 \* inv\_lamb.dot(beta\_mu) self.beta = s.dot(X.T.dot(y) + b0)

def predict(self, X): """ Prediction """

if self.fit\_intercept: X = add\_intercept(X)

return X.dot(self.beta) model = BayesianLinearRegression()

X\_train = X\_train.astype(float) X\_test = X\_test.astype(float)

model.fit\_map(X\_train,y\_train) y\_predictions = model.predict(X\_test)

df = pd.DataFrame(list(zip(y\_test,y\_predictions)),columns =['Actual', 'Predicted']) df.head(100)

def mean\_squared\_error(y\_true, y\_pred):

mse = np.square(np.subtract(y\_true,y\_pred)).mean() return mse

print(f"Mean squared error :: {mean\_squared\_error(y\_test, y\_predictions)}")

# Output:

