```
from google.colab import files
uploaded = files.upload()
```

Choose files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Train.csv to Train.csv
Saving Test.csv to Test.csv
Saving LR.csv to LR.csv
Saving RR.csv to RR.csv
Saving DT.csv to DT.csv
Saving data.csv to data.csv
Saving test_modified.csv to test_modified.csv
Saving train_modified.csv to train_modified.csv
```

```
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()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type
0	FDA15	9.30	Low Fat	0.016047	Dairy
1	DRC01	5.92	Regular	0.019278	Soft Drinks
2	FDN15	17.50	Low Fat	0.016760	Meat
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables
4	NCD19	8.93	Low Fat	0.000000	Household

```
trainDf.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
```

```
print(trainDf.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
```

```
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
combination["Item_Fat_Content"] = combination["Item_Fat_Content"].replace({"Low Fa
combination["Item_Fat_Content"] = combination["Item_Fat_Content"].replace({"low fa

perishable = ["Breads", "Breakfast", "Dairy", "Fruits and Vegetables", "Meat", "Sea
non_perishable = ["Baking Goods", "Canned", "Frozen Foods", "Hard Drinks", "Health
                  "Soft Drinks", "Snack Foods", "Starchy Foods", "Others"]

combination["Item_Type"] = combination["Item_Type"].replace(to_replace=perishable,
combination["Item_Type"] = combination["Item_Type"].replace(to_replace=non_perishal
combination["Item_Type"] = combination["Item_Type"].replace({"perishable": 0, "non_

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(



combination["Outlet_Type"] = combination["Outlet_Type"].replace({"Grocery Store":
                                                                 "Supermarket Type
                                                                 "Supermarket Type
                                                                 "Supermarket Type



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

```
testDfClean = combination[trainingSetIndex:]
```

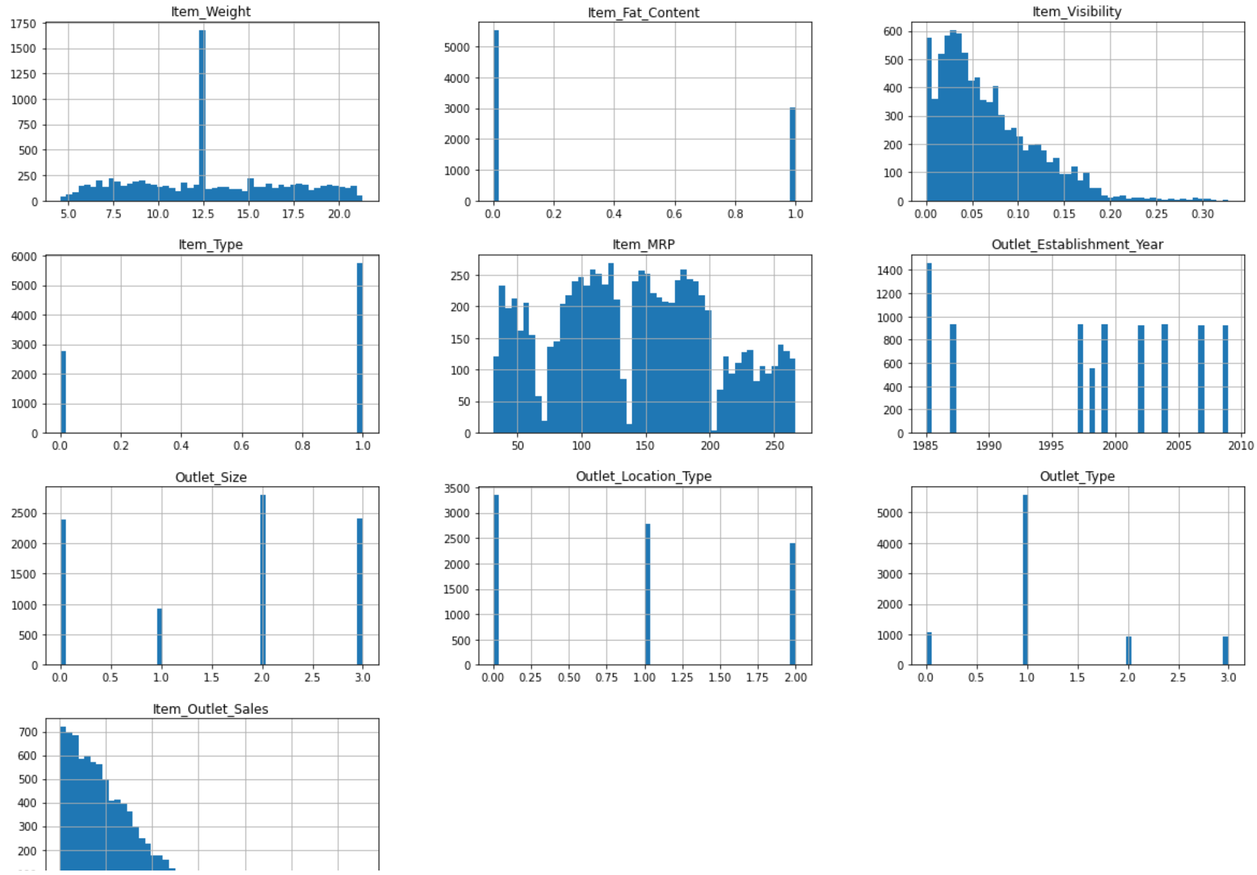
```
trainDfClean.head()
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_
0	9.30	0	0.016047	0	249.8092	
1	5.92	1	0.019278	1	48.2692	
2	17.50	0	0.016760	0	141.6180	
3	19.20	1	0.000000	0	182.0950	
4	8.93	0	0.000000	1	53.8614	

```
testDfClean.head()
```

	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_
0	20.750	0	0.007565	1	107.8622	
1	8.300	1	0.038428	0	87.3198	
2	14.600	0	0.099575	1	241.7538	
3	7.315	0	0.015388	1	155.0340	
4	12.600	1	0.118599	0	234.2300	

```
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
```

Linear Regression

```
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)
```

```
    1017.6778612869437
```

```
mean_squared_error(y_test, y_pred)
```

```
    1599790.077463071
```

```
df = pd.DataFrame(list(zip(y_test, y_pred)), columns =['Actual', 'Predicted'])
df.head(100)
```

	Actual	Predicted
0	1794.331	1849.892148
1	1794.331	1265.288073
2	1794.331	2289.287604
3	1794.331	2292.930401
4	1794.331	5121.067178
...	...	...
95	1794.331	898.477838
96	1794.331	1992.952332
97	1794.331	2839.656614
98	1794.331	2751.167112
99	1794.331	836.579946

Bayesian Linear Regression

```
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)
```

	Actual	Predicted
0	1794.331	1838.191471
1	1794.331	1271.368031
2	1794.331	2284.702917
3	1794.331	2296.661053
4	1794.331	5107.305066
...	...	...
95	1794.331	895.190439
96	1794.331	2002.151926
97	1794.331	2864.500194
98	1794.331	2750.002343
99	1794.331	818.497494

100 rows × 2 columns

```
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)}")
```

```
Mean squared error :: 1597765.6317229886
```

● ×