```
import pandas as pd
df1=pd.read_csv('/tmp/train.csv') #loading the train set
df2=pd.read_csv('/tmp/test.csv')  #loading the test set
```

```
df1.head(3)
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Typ
0	FDA15	9.30	Low Fat	0.016047	Dai
1	DRC01	5.92	Regular	0.019278	Soft Drink
2	FDN15	17.50	Low Fat	0.016760	Mea

```
# Checking Null values in Train Set
df1.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
```

```
# Checking Null values in Test Set
df2.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
```

```
#Filling the Null values in 'Item_Weight' column with the average weight
df1['Item_Weight'].fillna(value=df1['Item_Weight'].mean(),inplace=True)
df2['Item_Weight'].fillna(value=df2['Item_Weight'].mean(),inplace=True)

#Filling the Null values in 'Outlet_Size' column with "Unknown"
df1['Outlet_Size'].fillna(value='Unknown',inplace=True)
df2['Outlet_Size'].fillna(value='Unknown',inplace=True)
```

```
#Checking the Null values in Train set
df1.isnull().sum()
```

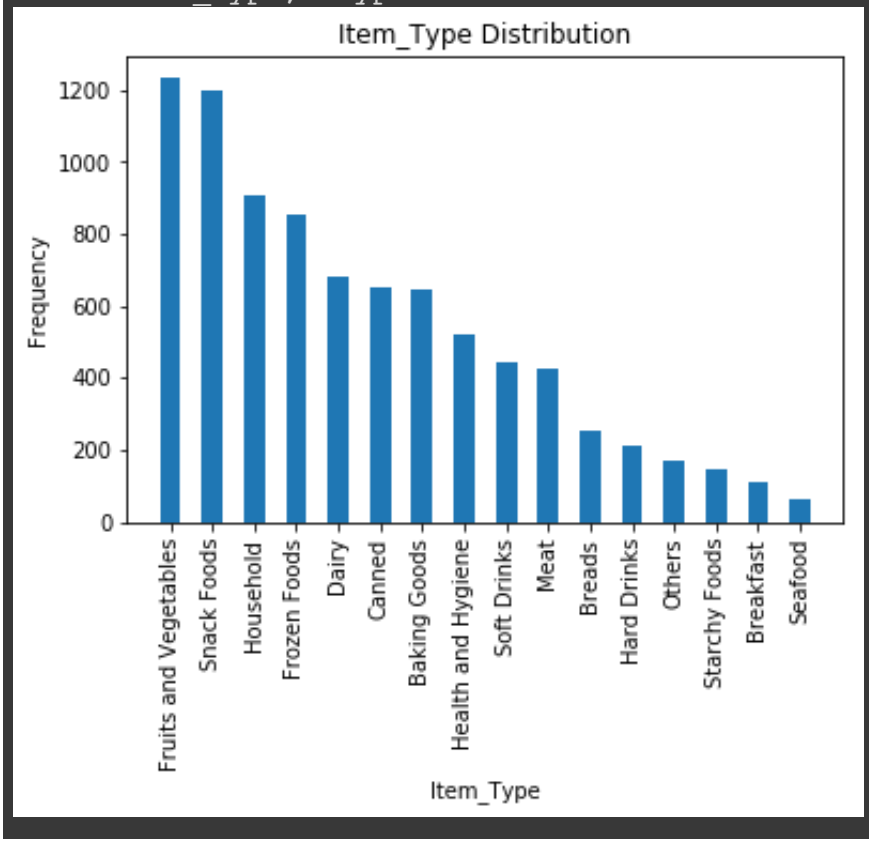
```
Item_Identifier            0
Item_Weight                0
Item_Fat_Content           0
Item_Visibility            0
Item_Type                  0
Item_MRP                   0
Outlet_Identifier          0
Outlet_Establishment_Year  0
Outlet_Size                0
Outlet_Location_Type       0
Outlet_Type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

```
#Checking the Null values in Test set
df2.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
dtype: int64
```

```
df1['Item_Type'].value_counts()
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Item_Type'].value_counts().index, df1['Item_Type'].value_counts(),
plt.title('Item_Type Distribution')
plt.xticks(rotation='vertical')
plt.xlabel('Item_Type')
plt.ylabel('Frequency')
print('Item_Type:\n',df1['Item_Type'].value_counts())
```

```
Item_Type:
 Fruits and Vegetables    1232
Snack Foods               1200
Household                  910
Frozen Foods               856
Dairy                      682
Canned                     649
Baking Goods               648
Health and Hygiene         520
Soft Drinks                445
Meat                       425
Breads                     251
Hard Drinks                214
Others                     169
Starchy Foods              148
Breakfast                  110
Seafood                     64
Name: Item_Type, dtype: int64
```



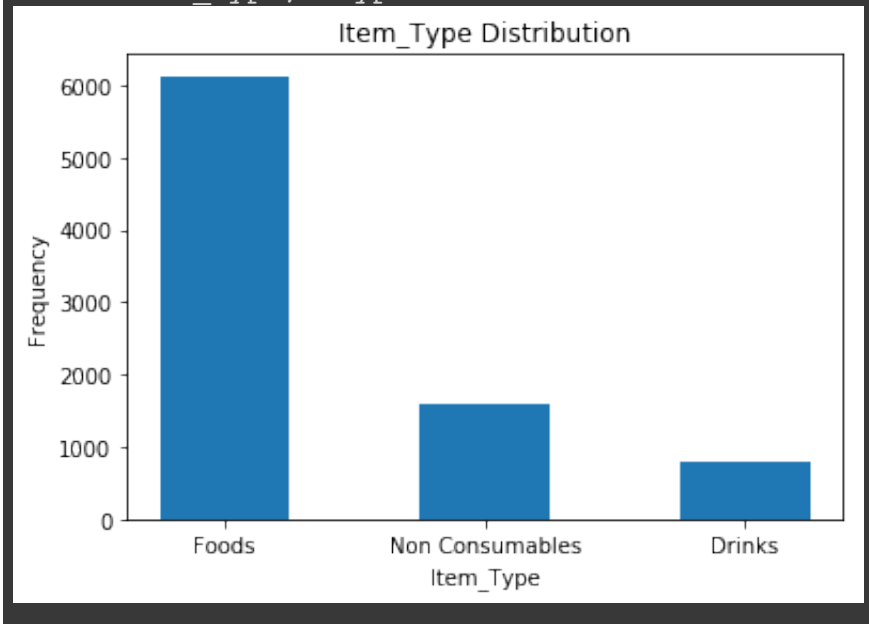
```
# Let's categorize 'Item_Type' into 'Non consumables', 'Drinks' and 'Foods' acc
def item_identify(cols):
  item_id=cols[0]
  item_type=cols[1]

  if item_id[:2] == 'NC':
    return 'Non Consumables'
  elif item_id[:2] == 'DR':
    return 'Drinks'
  else:
    return 'Foods'
```

```
df1['Item_Type']=df1[['Item_Identifier','Item_Type']].apply(item_identify,axis=
df2['Item_Type']=df2[['Item_Identifier','Item_Type']].apply(item_identify,axis=
```

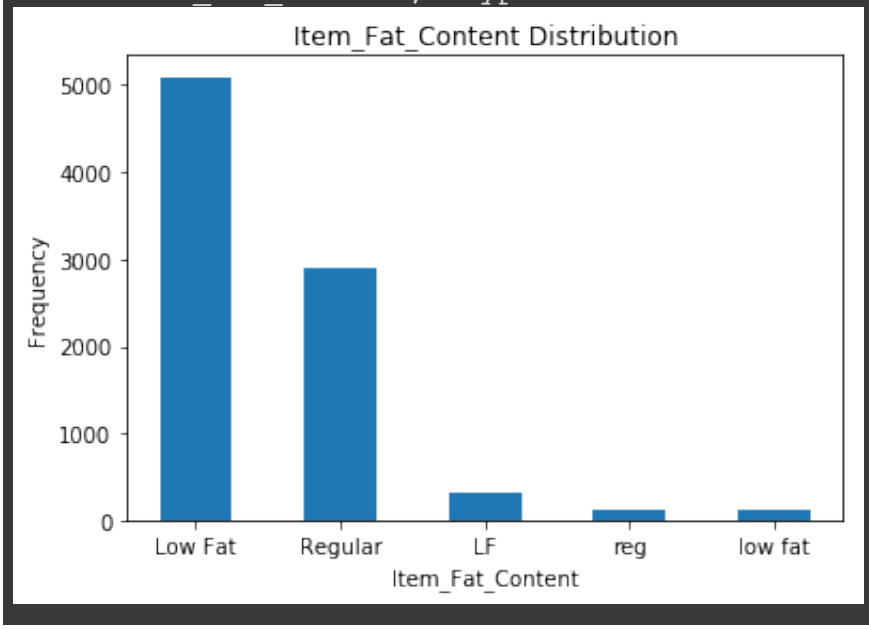
```
df1['Item_Type'].value_counts()
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Item_Type'].value_counts().index, df1['Item_Type'].value_counts(),
plt.title('Item_Type Distribution')
#plt.xticks(rotation='vertical')
plt.xlabel('Item_Type')
plt.ylabel('Frequency')
print('Item_Type:\n',df1['Item_Type'].value_counts())
```

```
Item_Type:
 Foods              6125
Non Consumables    1599
Drinks              799
Name: Item_Type, dtype: int64
```



```
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Item_Fat_Content'].value_counts().index, df1['Item_Fat_Content'].v
plt.title('Item_Fat_Content Distribution')
plt.xlabel('Item_Fat_Content')
plt.ylabel('Frequency')
print('Item_Fat_Content:\n',df1['Item_Fat_Content'].value_counts())
#df1['Item_Fat_Content'].value_counts()
```

```
Item_Fat_Content:
 Low Fat    5089
Regular    2889
LF          316
reg         117
low fat     112
Name: Item_Fat_Content, dtype: int64
```



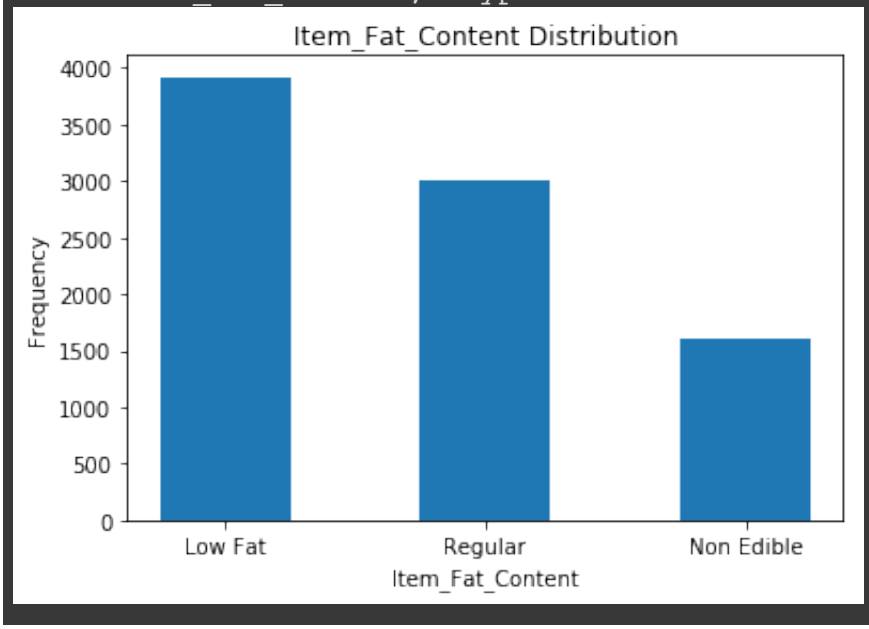

```
# From the above graph we can see that there are actually two classes in 'Item_
def item_fat(cols):
  fat=cols[0]
  typ=cols[1]

  if (fat=='Low Fat' or fat=='LF' or fat=='low fat') and (typ=='Foods' or typ==
    return 'Low Fat'
  elif (fat=='Regular' or fat=='reg') and (typ=='Foods' or typ=='Drinks'):
    return 'Regular'
  else:
    return 'Non Edible'
```

```
df1['Item_Fat_Content']=df1[['Item_Fat_Content','Item_Type']].apply(item_fat,ax
df2['Item_Fat_Content']=df2[['Item_Fat_Content','Item_Type']].apply(item_fat,ax
```

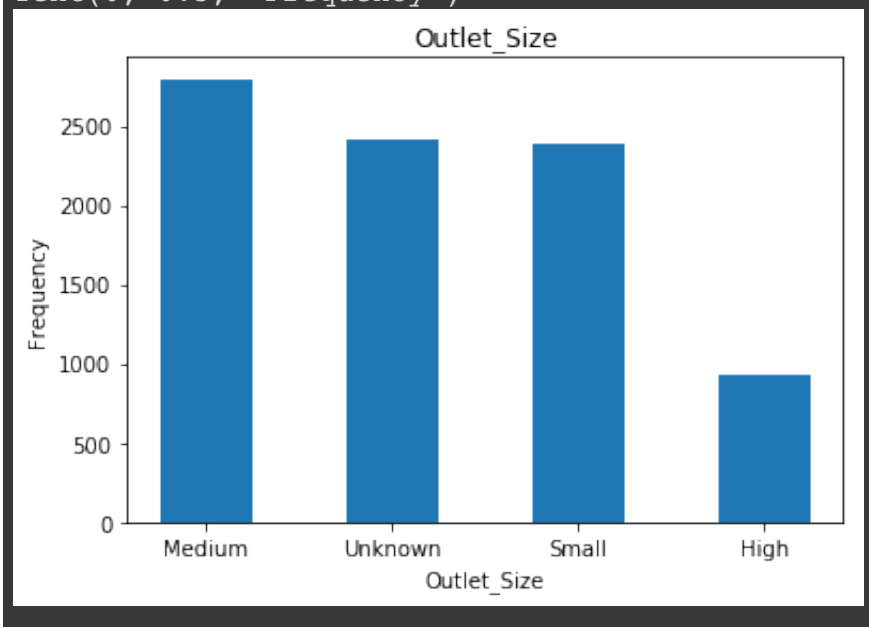
```
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Item_Fat_Content'].unique(), df1['Item_Fat_Content'].value_counts(
plt.title('Item_Fat_Content Distribution')
plt.xlabel('Item_Fat_Content')
plt.ylabel('Frequency')
print('Item_Fat_Content:\n',df1['Item_Fat_Content'].value_counts())
```

```
Item_Fat_Content:
 Low Fat       3918
Regular       3006
Non Edible    1599
Name: Item_Fat_Content, dtype: int64
```



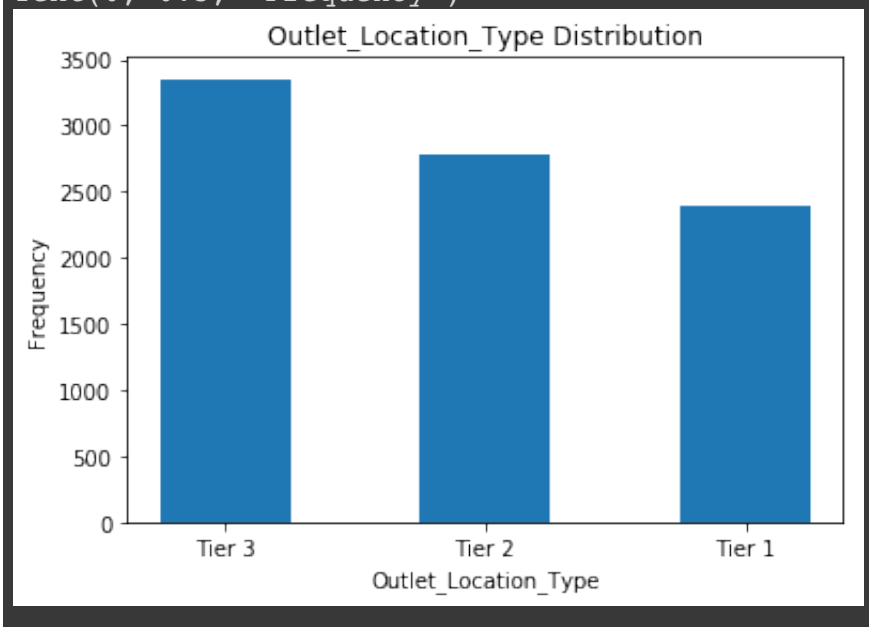
```
df1['Outlet_Size'].value_counts()
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Outlet_Size'].value_counts().index, df1['Outlet_Size'].value_count
plt.title('Outlet_Size')
#plt.xticks(rotation='vertical')
plt.xlabel('Outlet_Size')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



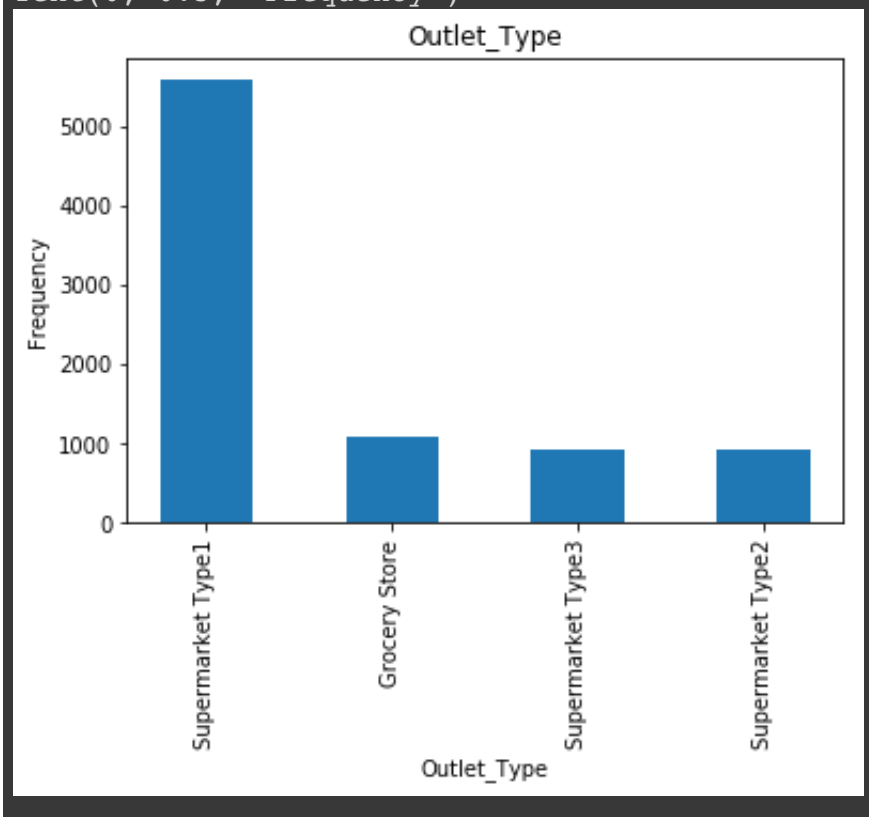
```
df1['Outlet_Location_Type'].value_counts()
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Outlet_Location_Type'].value_counts().index, df1['Outlet_Location_
plt.title('Outlet_Location_Type Distribution')
#plt.xticks(rotation='vertical')
plt.xlabel('Outlet_Location_Type')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```



```
df1['Outlet_Type'].value_counts()
import numpy as np
import matplotlib.pyplot as plt
plt.bar(df1['Outlet_Type'].value_counts().index, df1['Outlet_Type'].value_count
plt.title('Outlet_Type')
plt.xticks(rotation='vertical')
plt.xlabel('Outlet_Type')
plt.ylabel('Frequency')
```

```
Text(0, 0.5, 'Frequency')
```




```
df1['Item_Visibility'].value_counts().head(3)
```

```
0.000000    526
0.076975      3
0.041283      2
Name: Item_Visibility, dtype: int64
```

```
#The column 'Item_Visibility' has lots of 0 values. Let's fill this values with
df1['Item_Visibility'].mask(df1['Item_Visibility']== 0,df1['Item_Visibility'].m
df2['Item_Visibility'].mask(df2['Item_Visibility']== 0,df2['Item_Visibility'].m
```

```
#Let's add a new feature that would have the number of years the outlet has bee
def num_years(col):
  return 2013-col
```

```
df1['Years_of_Operation']=df1['Outlet_Establishment_Year'].apply(num_years)
df2['Years_of_Operation']=df2['Outlet_Establishment_Year'].apply(num_years)
```

```
# One hot encoding the categorical variables in both train and test set
item_fat_content=pd.get_dummies(df1['Item_Fat_Content'])
item_type=pd.get_dummies(df1['Item_Type'])
outlet_size=pd.get_dummies(df1['Outlet_Size'])
outlet_location_type=pd.get_dummies(df1['Outlet_Location_Type'])
output_type=pd.get_dummies(df1['Outlet_Type'])

item_fat_content_test=pd.get_dummies(df2['Item_Fat_Content'])
item_type_test=pd.get_dummies(df2['Item_Type'])
outlet_size_test=pd.get_dummies(df2['Outlet_Size'])
outlet_location_type_test=pd.get_dummies(df2['Outlet_Location_Type'])
output_type_test=pd.get_dummies(df2['Outlet_Type'])
```

```
train=df1
train=pd.concat([train,item_fat_content,item_type,outlet_size,outlet_location_t
train.drop(['Item_Identifier','Item_Fat_Content','Item_Type','Outlet_Identifier
       'Outlet_Type'],axis=1,inplace=True)
test=df2
test=pd.concat([test,item_fat_content_test,item_type_test,outlet_size_test,outl
test.drop(['Item_Identifier','Item_Fat_Content','Item_Type','Outlet_Identifier'
       'Outlet_Type'],axis=1,inplace=True)
```

```
x=train.drop(['Item_Outlet_Sales'],axis=1)
y=train['Item_Outlet_Sales']
x_test=test
```

```
#Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_x=StandardScaler()
x=sc_x.fit_transform(x)

x_test=sc_x.fit_transform(x_test)
```

```
from sklearn.model_selection import train_test_split
x_train,x_val,y_train,y_val=train_test_split(x,y,test_size=0.1,random_state=42)
```

Linear Regression

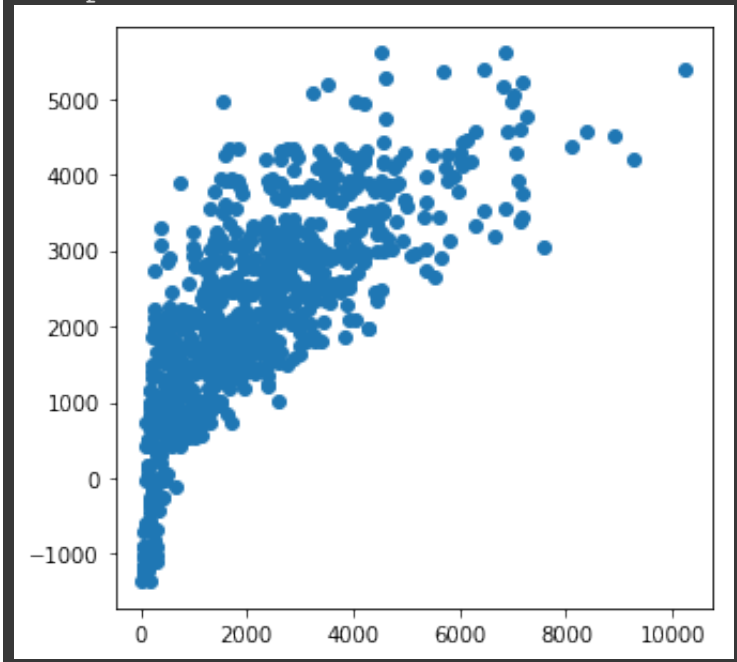
```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
normalize=False)
```

```
predictions=lm.predict(x_val)
```

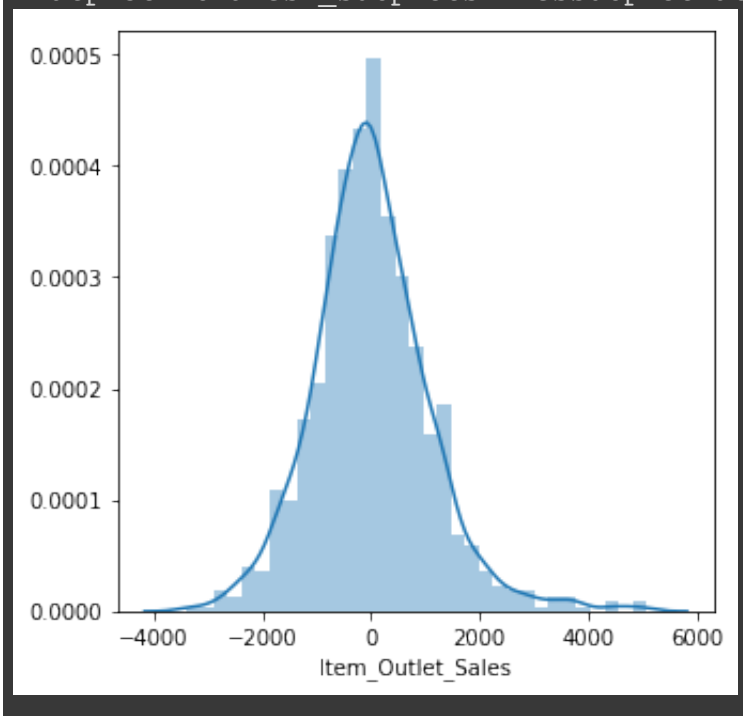
```
plt.scatter(y_val,predictions)
```

```
<matplotlib.collections.PathCollection at 0x7f71d324ae10>
```



```
import seaborn as sns
sns.distplot((y_val-predictions))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71d326f4e0>
```




```
import numpy as np
from sklearn import metrics
print('Mean Absolute Error: ',metrics.mean_absolute_error(y_val,predictions))
print('Mean Squared Error: ',metrics.mean_squared_error(y_val,predictions))
print('Root Mean Squared Error: ',np.sqrt(metrics.mean_squared_error(y_val,prec
print('Explained Variance Score: ',metrics.explained_variance_score(y_val,predi
```

```
Mean Absolute Error:  805.161700156959
Mean Squared Error:  1179555.069926781
Root Mean Squared Error:  1086.0732341452767
Explained Variance Score:  0.5781432231758533
```

Random Forest

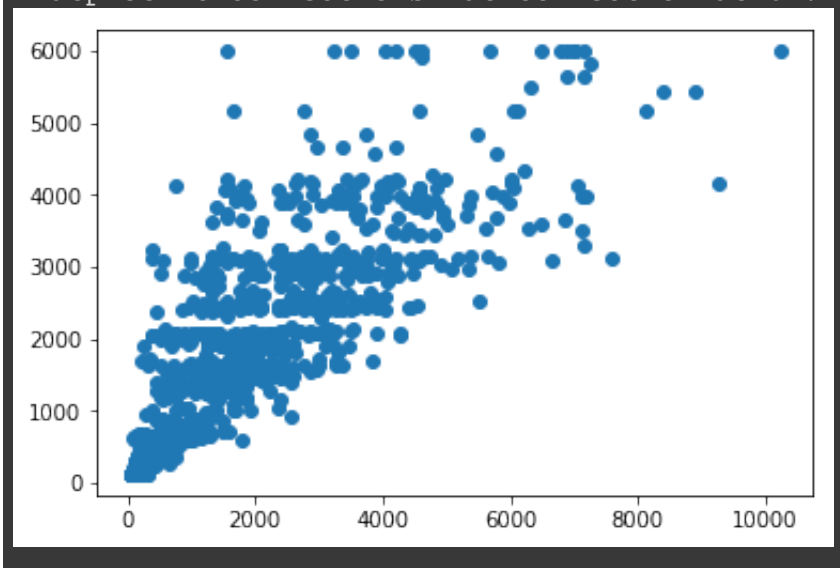
```
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(n_estimators=400,max_depth=6, min_samples_leaf=76,n_jo
rf.fit(x_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=6,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=76, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=400,
    n_jobs=4,
                      oob_score=False, random_state=None, verbose=0,
                      warm_start=False)
```

```
predictions1=rf.predict(x_val)
```

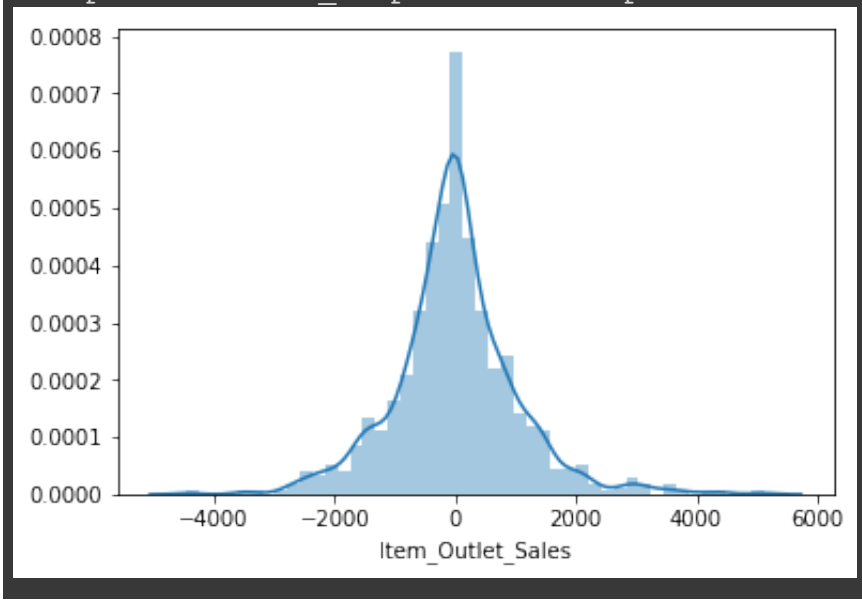
```
plt.scatter(y_val,predictions1)
```

```
<matplotlib.collections.PathCollection at 0x7ff17b145d30>
```



```
import seaborn as sns
sns.distplot((y_val-predictions1))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff17ad733c8>
```




```
import numpy as np
from sklearn import metrics
print('Mean Absolute Error: ',metrics.mean_absolute_error(y_val,predictions1))
print('Mean Squared Error: ',metrics.mean_squared_error(y_val,predictions1))
print('Root Mean Squared Error: ',np.sqrt(metrics.mean_squared_error(y_val,prec
print('Explained Variance Score: ',metrics.explained_variance_score(y_val,predi
```

```
Mean Absolute Error:  721.9482692470027
Mean Squared Error:  1060359.6575380685
Root Mean Squared Error:  1029.7376644262695
Explained Variance Score:  0.6207417638502686
```

```
#Grid Search
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    #'bootstrap': [True],
    #'max_depth': range(1,15,1),
    #'min_samples_leaf': range(70,80,1),
    #'min_samples_split': [8, 10, 12],
    'n_estimators': range(100,500,100)
}

# Create a based model
rf = RandomForestRegressor()

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                           cv = 10, n_jobs = 4, verbose = 2,scoring='neg_mean_ab
# Fit the grid search to the data
grid_search.fit(x, y)

grid_search.best_params_,grid_search.best_score_
```

```
Fitting 10 folds for each of 4 candidates, totalling 40 fits
[Parallel(n_jobs=4)]: Using backend LokyBackend with 4 concurrent workers.
/usr/local/lib/python3.6/dist-packages/joblib/externals/loky/process_execut
  "timeout or by a memory leak.", UserWarning
[Parallel(n_jobs=4)]: Done  33 tasks      | elapsed:  3.5min
[Parallel(n_jobs=4)]: Done  40 out of  40 | elapsed:  4.4min finished
({'n_estimators': 400}, -797.8347339432124)
```

XGBoost

```
import xgboost as xgb
data_dmatrix = xgb.DMatrix(data=x,label=y)
xg_reg = xgb.XGBRegressor(objective ='reg:squarederror',
                          learning_rate =0.1,
                          n_estimators=41,
                          max_depth=3,
                          min_child_weight=5,
                          gamma=0,
                          subsample=0.75,
                          colsample_bytree=0.95,
                          nthread=4,
                          scale_pos_weight=1,
                          reg_alpha=0.021,
                          seed=42,
                          )
xg_reg.fit(x_train,y_train)
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
  data.base is not None and isinstance(data, np.ndarray) \
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=0.95, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=5, missing=None,
n_estimators=41,
             n_jobs=1, nthread=4, objective='reg:squarederror',
random_state=0,
             reg_alpha=0.021, reg_lambda=1, scale_pos_weight=1, seed=42,
             silent=None, subsample=0.75, verbosity=1)
```

```
predictions2 = xg_reg.predict(x_val)
```

```
import numpy as np
from sklearn import metrics
print('Mean Absolute Error: ',metrics.mean_absolute_error(y_val,predictions2))
print('Mean Squared Error: ',metrics.mean_squared_error(y_val,predictions2))
print('Root Mean Squared Error: ',np.sqrt(metrics.mean_squared_error(y_val,prec
print('Explained Variance Score: ',metrics.explained_variance_score(y_val,predi
```

```
Mean Absolute Error:  720.3817910920656
Mean Squared Error:  1059996.284783057
Root Mean Squared Error:  1029.561209828273
Explained Variance Score:  0.6210769021550897
```

Grid Search for best parameters

```
from sklearn.model_selection import GridSearchCV

#Tune max_depth and min_child_weight
param_test1 = {
 'max_depth':range(2,4,1),
 'min_child_weight':range(3,6,1)
}
gsearch1 = GridSearchCV(estimator = xg_reg,
                        param_grid = param_test1,
                        scoring='neg_mean_absolute_error',
                        n_jobs=4,iid=False, cv=10)
gsearch1.fit(x,y)
gsearch1.best_params_, gsearch1.best_score_
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
  data.base is not None and isinstance(data, np.ndarray) \
({'max_depth': 3, 'min_child_weight': 5}, -757.4700216462467)
```

```
#Tuning Gamma
param_test2 = {
 'gamma':[i/10.0 for i in range(0,10)]
}
gsearch2 = GridSearchCV(estimator = xg_reg,
                        param_grid = param_test2,
                        scoring='neg_mean_absolute_error',
                        n_jobs=4,iid=False, cv=10)

gsearch2.fit(x,y)
gsearch2.best_params_, gsearch2.best_score_
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
  data.base is not None and isinstance(data, np.ndarray) \
({'gamma': 0.0}, -757.4700216462467)
```

```
#Tune subsample and colsample_bytree
param_test3 = {
 'subsample':[i/100.0 for i in range(75,100,5)],
 'colsample_bytree':[i/100.0 for i in range(75,100,5)]
}

gsearch3 = GridSearchCV(estimator = xg_reg,
                        param_grid = param_test3,
                        scoring='neg_mean_absolute_error',
                        n_jobs=4,iid=False, cv=10)
gsearch3.fit(x,y)
gsearch3.best_params_, gsearch3.best_score_
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
  data.base is not None and isinstance(data, np.ndarray) \
({'colsample_bytree': 0.95, 'subsample': 0.75}, -757.0455675274022)
```

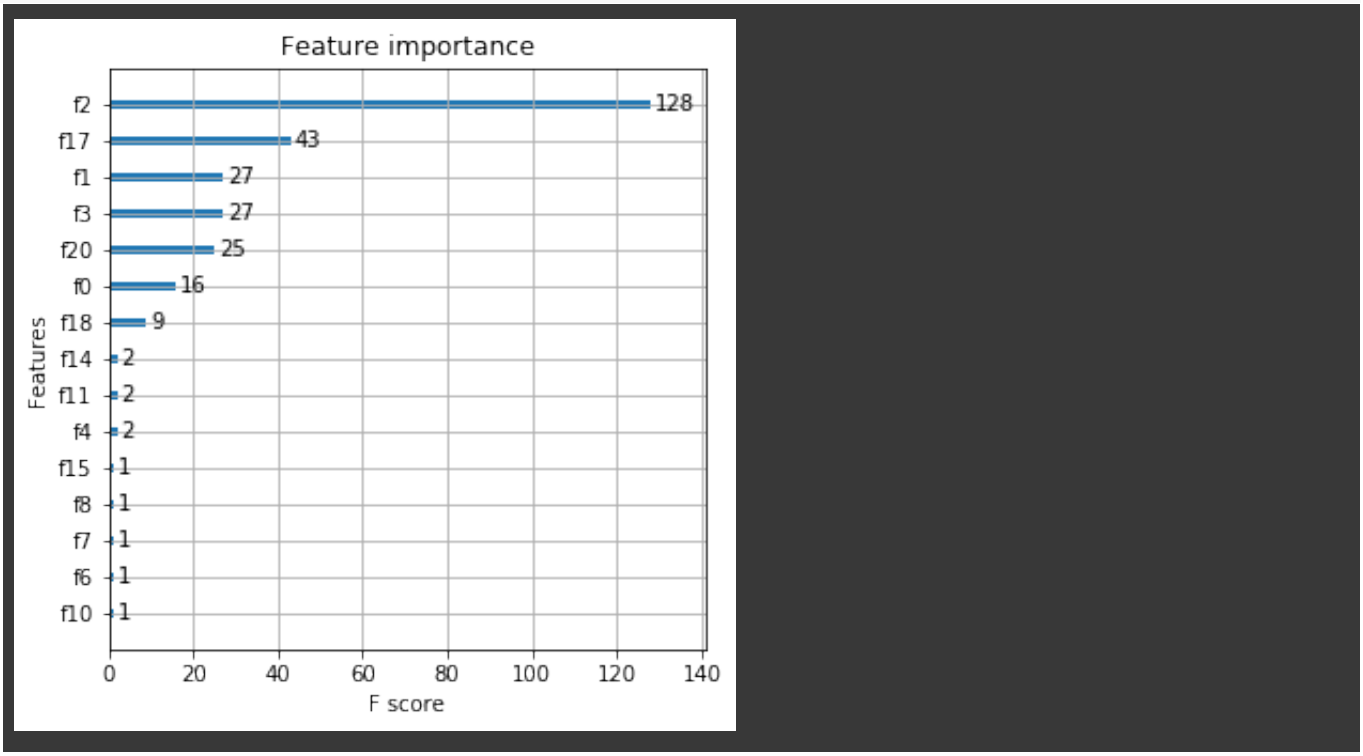
```
# Tuning Regularization Parameters
param_test4 = {
 'reg_alpha':[0.011,0.012,0.013,0.014,0.015,0.016,0.017,0.018,0.019,0.02,0.021,
}
gsearch4 = GridSearchCV(estimator = xg_reg,
                        param_grid = param_test4,
                        scoring='neg_mean_absolute_error',
                        n_jobs=4,iid=False, cv=10)
gsearch4.fit(x,y)
gsearch4.best_params_, gsearch4.best_score_
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
  data.base is not None and isinstance(data, np.ndarray) \
({'reg_alpha': 0.021}, -755.5158996159646)
```

```
# Tuning No. of trees
param_test5 = {
 'n_estimators':range(30,50,1)
}
gsearch5 = GridSearchCV(estimator = xg_reg,
                        param_grid = param_test5,
                        scoring='neg_mean_absolute_error',
                        n_jobs=4,iid=False, cv=10)
gsearch5.fit(x,y)
gsearch5.best_params_, gsearch5.best_score_
```

```
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:587: FutureWarning:
  if getattr(data, 'base', None) is not None and \
/usr/local/lib/python3.6/dist-packages/xgboost/core.py:588: FutureWarning:
  data.base is not None and isinstance(data, np.ndarray) \
({'n_estimators': 41}, -755.5159016995146)
```

```
xgb.plot_importance(xg_reg)
plt.rcParams['figure.figsize'] = [5, 5]
plt.show()
```




```
#Generating the Sales for the Test set and saving in a .csv file
pred_test = xg_reg.predict(x_test)
out_df = pd.DataFrame({'Item_Identifier':df2['Item_Identifier'].values,'Outlet_
out_df['Item_Outlet_Sales'] = pred_test
out_df.to_csv('submission.csv', index=False)
```