BIGMART SUPERMARKET Sales Prediction.

```
pip install pyforest
```

```
from pyforest import*
lazy_imports()
```

```
import warnings
warnings.filterwarnings("ignore")
```

```
Train_data=pd.read_csv('bigmart_train.csv')
Test_data=pd.read_csv('bigmart_test.csv')
```

```
Train_data.head()
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_M
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.80
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.26
2	FDN15	17.50	Low Fat	0.016760	Meat	141.61
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.09
4	NCD19	8.93	Low Fat	0.000000	Household	53.86

```
Test_data.head()
```

Exploratory Data Analysis

```
print('Train_data:',Train_data.shape)
print('Test_data:',Test_data.shape)
```

```
Train_data.describe().T
```

```
Train_data.isnull().sum()
```

```
Test_data.isnull().sum()
```

```
Train_data['source'] = 'train'
Test_data['source'] = 'test'
df=pd.concat([Train_data,Test_data], ignore_index=True)
```

```
df.head()
```

```
df.tail()
```

```
df.isnull().sum()
```

```
df.shape
```

```
for i in Train_data.describe().columns:
  sns.distplot(Train_data[i].dropna())
  plt.show()
```

```
for i in Train_data.describe().columns:
  sns.boxplot(Train_data[i].dropna())
  plt.show()
```

```
plt.figure(figsize=(15,10))
sns.countplot(Train_data.Item_Type)
plt.xticks(rotation=90)
```

```
Train_data.Item_Type.value_counts()
```

Distribution of the outlet_Size

```
plt.figure(figsize=(10,8))
sns.countplot(Train_data.Outlet_Size)
```

```
Train_data.Outlet_Size.value_counts()
```

Distribution of the Outlet_Location_Type

```
plt.figure(figsize=(10,8))
sns.countplot(Train_data.Outlet_Location_Type)
```

```
Train_data.Outlet_Location_Type.value_counts()
```

Distribution of the Outlet_Type

```
plt.figure(figsize=(10,8))
sns.countplot(Train_data.Outlet_Type)
plt.xticks(rotation=10)
```

```
Train_data.Outlet_Type.value_counts()
```

Item_Weight and Item_Outlet_Sales Analysis

```
plt.figure(figsize=(13,9))
plt.xlabel('Item_Weight')
plt.ylabel('Item_Outlet_Sales')
plt.title('Item_Weight and Item_Outlet_Sales Analysis')
sns.scatterplot(x='Item_Weight', y='Item_Outlet_Sales',hue='Item_Type', size='Item_Weight',data=Train_data)
```

Item_Visibility and Maximum Retail Price

```
plt.figure(figsize=(12,7))
plt.xlabel('Item_Visibility')
plt.ylabel('Maximum Retail Price')
plt.title('Item_Visibility and Maximum Retail Price')
plt.plot(Train_data.Item_Visibility, Train_data.Item_MRP, ".",alpha = 0.3)
```

Impact of Outlet_Type on Item_Outlet_Sales

```
Item_Type_pivot = \
Train_data.pivot_table(index='Outlet_Type', values="Item_Outlet_Sales", aggfunc=np.median)

Item_Type_pivot.plot(kind='bar',color='brown',figsize=(12,7))
plt.xlabel('Outlet_Type')
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Outlet_Type on Item_Outlet_Sales")
```

```
plt.xticks(rotation=0)
```

Impact of Item_Fat_Content on Item_Outlet_Sales

```
Item_Type_pivot = \
Train_data.pivot_table(index='Item_Fat_Content', values="Item_Outlet_Sales", aggfunc=np.median)

Item_Type_pivot.plot(kind='bar',color='blue',figsize=(12,7))
plt.xlabel('Item_Fat_Content')
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Item_Fat_Content on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```

```
df['Item_Fat_Content'].value_counts()
```

```
df['Item_Fat_Content'] = df['Item_Fat_Content'].replace({'LF':'Low Fat','reg':'Regular','low fat':'Low Fat',})
```

```
df['Item_Fat_Content'].value_counts()
```

```
Train_data['Item_Fat_Content'] = Train_data['Item_Fat_Content'].replace({'LF':'Low Fat','reg':'Regular','low fat'
```

```
Item_Fat_Content_pivot =\
Train_data.pivot_table(index='Item_Fat_Content',values='Item_Outlet_Sales',aggfunc=np.median)

Item_Fat_Content_pivot.plot(kind='bar',color='blue',figsize=(12,7))
plt.xlabel('Item_Fat_Content')
plt.ylabel("Item_Outlet_Sales")
plt.title("Impact of Item_Fat_Content on Item_Outlet_Sales")
plt.xticks(rotation=0)
plt.show()
```

```
Train_data.corr()
```

Correlation between different attributes

```
plt.figure(figsize=(35,15))
sns.heatmap(Train_data.corr(),vmax=1, square=True,annot=True, cmap='viridis')
plt.title('Correlation between different attributes')
plt.show()
```

Feature Engineering , Selection and Transformation

Treating The Missing Values

Item_Weight

From the boxplot we plotted at the beginning, we noticed that the **item_weight** column is approximately **normal** and it is therefore helpful to replace the missing values with the **Mean** of the column.

```
df['Item_Weight'].mean() #we will replace the NaN values with this mean
```

```
df['Item_Weight'].fillna(df['Item_Weight'].mean(), inplace=True) #missing values have been replaced with the mear
```

Outlet_Size

We will replace the missing values in **Outlet_Size** with the item that appears frequently, in this case **Meduim**.

```
df['Outlet_Size'].value_counts()
```

```
df['Outlet_Size'].fillna('Medium', inplace=True)
```

```
df.isnull().sum() #now we do not have any null values in Outlet_Size
```

```
Train_data.shape
```

```
Test_data.shape
```

```
df.shape
```

Item_Visibility

```
df[df['Item_Visibility']==0]['Item_Visibility'].count()
```

```
df['Item_Visibility'].fillna(df['Item_Visibility'].median(), inplace=True)
```

Outlet_Years

```
df['Outlet_Establishment_Year'].value_counts()
```

```
df['Outlet_Years'] = 2009-df['Outlet_Establishment_Year']
df['Outlet_Years'].describe()
```

Item_Type

```
df['Item_Type'].value_counts()
```

The item types are either Food, Drinks or Non-Consumables

```
df['Item_Identifier'].value_counts()
```

A closer look at each of the **Item_Identifier** shows that they with either **"FD", "DR"(Drinks), "NC"(Non-Consumables)**

For us to do a better analysis, we will be creating 3 categories as pointed out instead of the already existing 16 categories.

```
#Changing only the first 2 characters (i.e. the category ID)

df['New_Item_type'] = df['Item_Identifier'].apply(lambda x: x[0:2])
```

```
#Rename them to more intuitive  categories:

df['New_Item_type'] = df['New_Item_type'].map({'FD':'Food','NC':'Non-Consumable','DR':'Drinks'})
df['New_Item_type'].value_counts()
```

If a product is non-consumable then why associate a fat-content to them? We will get rid of this.

```
df.loc[df['New_Item_type']=='Non-Consumable','Item_Fat_Content']= "Non-Edible"
df['Item_Fat_Content'].value_counts()
```

Under normal circumstance, if a product is more visible, then it's likely it will be getting higher sales. We can based on that hypothesis and create importance given to a product in a given store according to the mean of significance given to the same product in all other stores.

```
item_visib_avg = df.pivot_table(values='Item_Visibility', index='Item_Identifier')
```

```
item_visib_avg
```

```
function = lambda x: x['Item_Visibility']/item_visib_avg['Item_Visibility'][item_visib_avg.index == x['Item_Ident
df['item_visib_avg'] = df.apply(function,axis=1).astype(float)
```

```
df.head()
```

Dealing with our Categorical Variables

Label Encoder We will be converting all categorical variables into numeric types (Values of 0 or 1) using the LabelEncoder function since we cannot build model on them.

```
from sklearn.preprocessing import LabelEncoder
label = LabelEncoder()

#New variable for outlet
df['Outlet'] = label.fit_transform(df['Outlet_Identifier'])
varib = ['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','New_Item_type','Outlet_Type','Outlet']
for i in varib:
  df[i] = label.fit_transform(df[i])
```

```
df.head()
```

```
#Dummy Variables:

df = pd.get_dummies(df, columns =['Item_Fat_Content','Outlet_Location_Type','Outlet_Size','New_Item_type','Outlet
df.dtypes
```

Model Building

```
df.drop(['Item_Type','Outlet_Establishment_Year'],axis=1, inplace=True)
```

```
train = df.loc[df['source'] == 'train']
test = df.loc[df['source'] == 'test']
```

```
train.drop(['source'],axis=1,inplace=True)
```

```
test.drop(['Item_Outlet_Sales','source'],axis=1,inplace=True)
```

```
X_train = train.drop(['Item_Outlet_Sales','Item_Identifier','Outlet_Identifier'],axis=1)
y_train = train['Item_Outlet_Sales']
X_test = test.drop(['Item_Identifier','Outlet_Identifier'],axis=1).copy()
```

Linear Regression

```
from sklearn.linear_model import LinearRegression

lr = LinearRegression(normalize=True)

lr.fit(X_train,y_train)
```

```
lr_pred = lr.predict(X_test)
```

```
lr_pred
```

```
lr_accuracy = round(lr.score(X_train,y_train)*100)
lr_accuracy
```

DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
```

```
tree = DecisionTreeRegressor(max_depth=15, min_samples_leaf=100)

tree.fit(X_train,y_train)
tree_pred = tree.predict(X_test)
```

```
tree_pred
```

```
tree_accuracy = round(tree.score(X_train,y_train)*100)
tree_accuracy
```

RandomForestRegressor

```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(n_estimators=400,max_depth=6,min_samples_leaf=100,n_jobs=4)

rf.fit(X_train,y_train)

rf_accuracy = round(rf.score(X_train,y_train)*100)

rf_accuracy
```

XGBoost Regressor

```
from xgboost import XGBRegressor

model = XGBRegressor(n_estimators = 100, learning_rate=0.05)
model.fit(X_train, y_train)
```

```
y_pred = model.predict(X_test)
y_pred
```

```
model.score(X_train, y_train)*100
```

NB: Output might not be same all the time you run the code. It may go up or down. In the real world scenario, we save the model as soon we get the highest accuracy.

---------- END OF PROJECT ----------

✓ 0s completed at 12:50