In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
```

In [2]:
```
# import dataset
df = pd.read_csv('Train-Set.csv')
df.head()
```

Out[2]:

	ProductID	Weight	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType	Ou
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	4
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	7
4	NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	9

Data Preprocessing Part 1

In [3]:
```
#Check ther number of unique value
df.select_dtypes(include='object').nunique()
```

Out[3]:
```
ProductID       1559
FatContent         5
ProductType       16
OutletID          10
OutletSize         3
LocationType       3
OutletType         4
dtype: int64
```

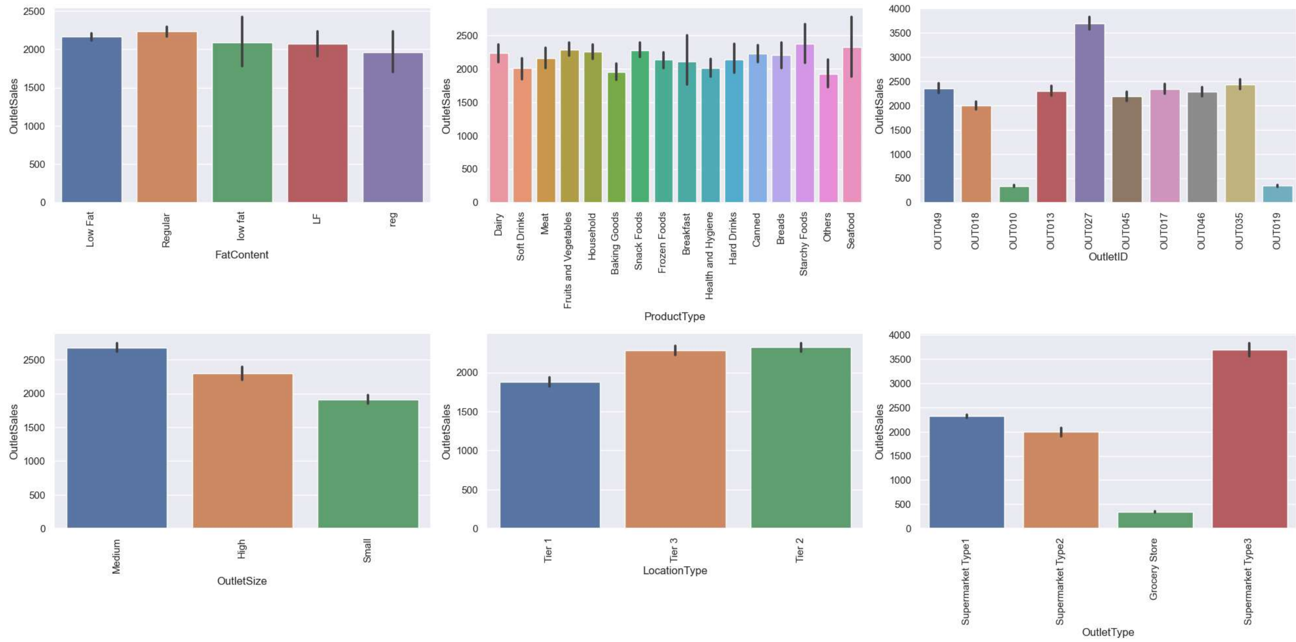
In [4]:
```
# Drop ProductID because its unnecessary
df.drop(columns='ProductID', inplace=True)
df.head()
```

Out[4]:

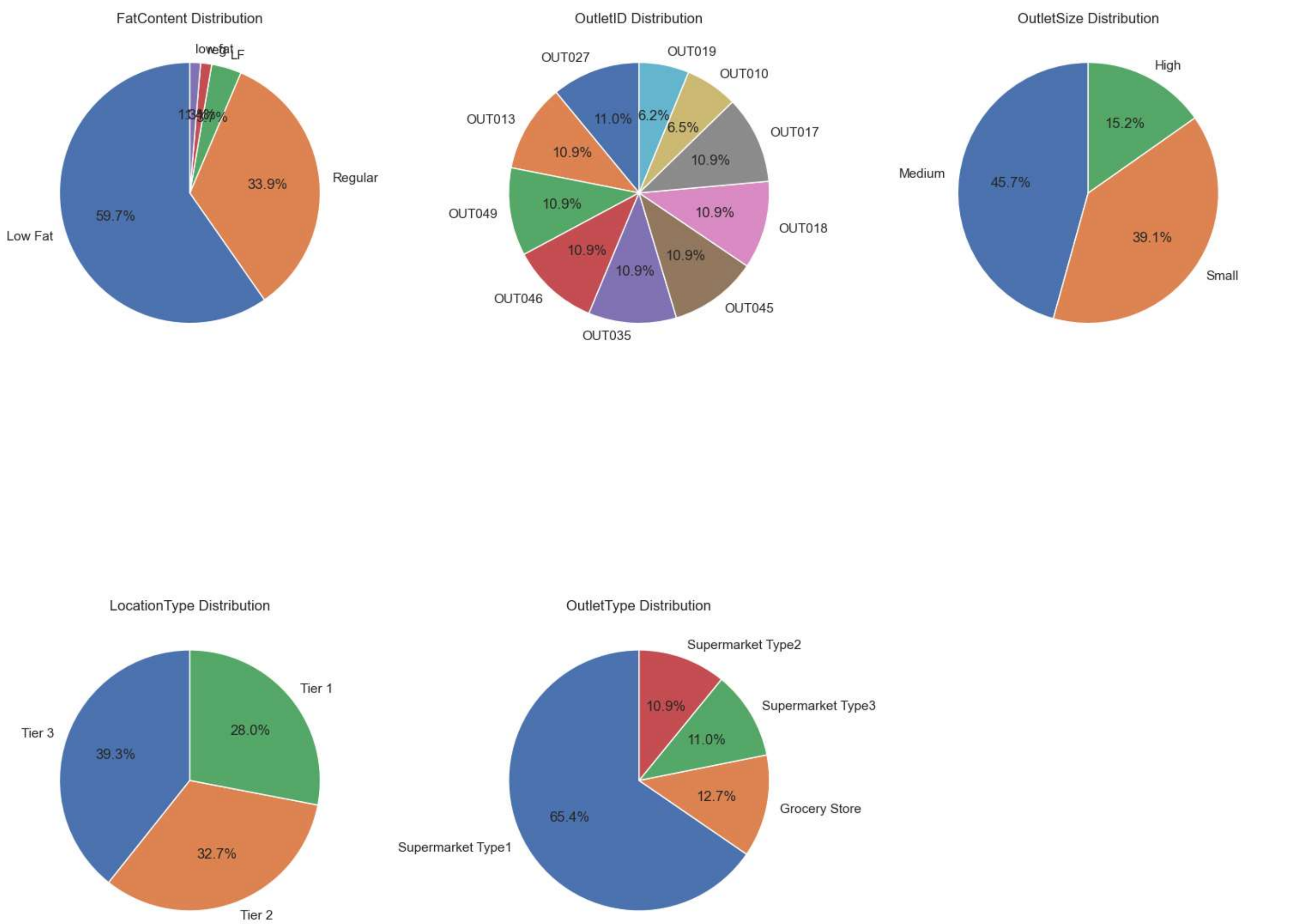
	Weight	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType	OutletSales
0	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800
4	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

Exploratory Data Analysis

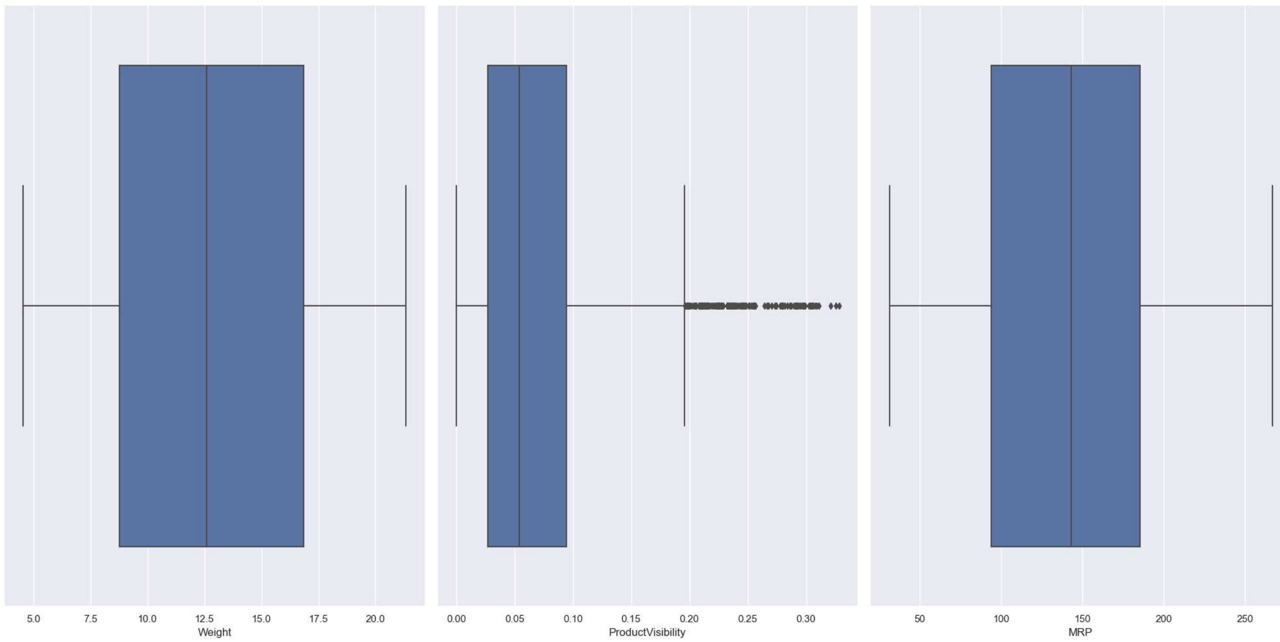
In [5]:
```python
# list of categorical variables to plot
cat_vars = ['FatContent', 'ProductType', 'OutletID', 'OutletSize', 'LocationType',
            'OutletType']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(20, 10))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.barplot(x=var, y='OutletSales', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



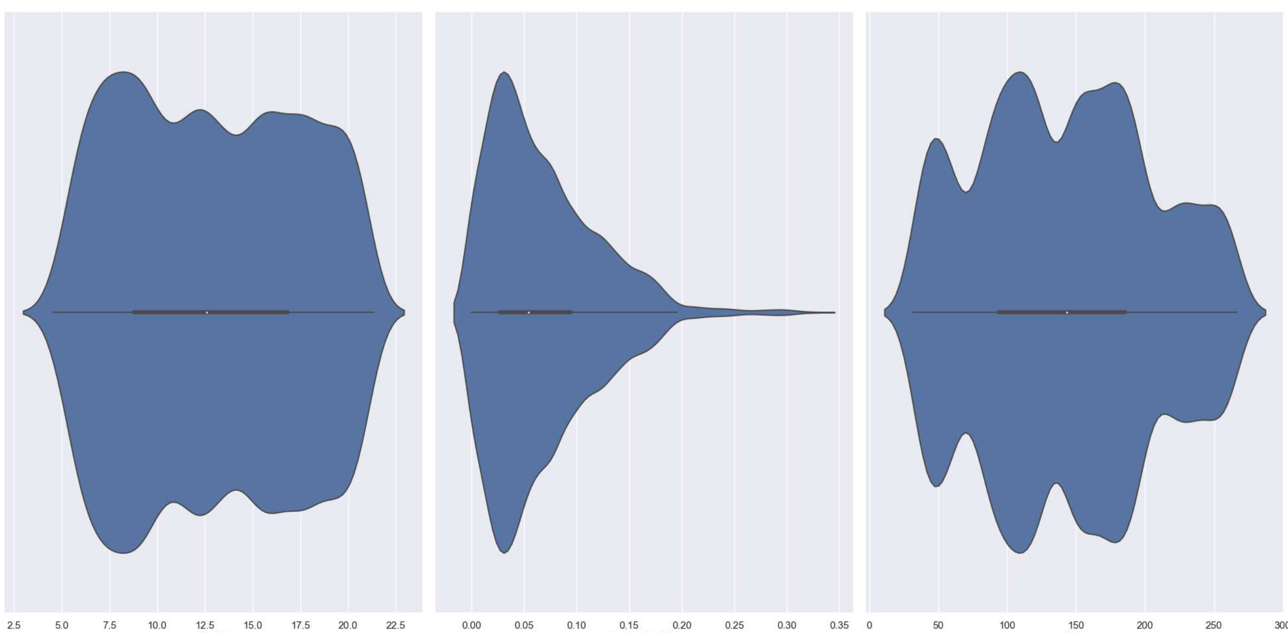
In [9]:
```python
cat_vars = ['FatContent', 'OutletID', 'OutletSize', 'LocationType',
            'OutletType']

# create a figure and axes
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 15))

# create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)

        # set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# adjust spacing between subplots
fig.tight_layout()
fig.delaxes(axs[1][2])

# show the plot
plt.show()
```



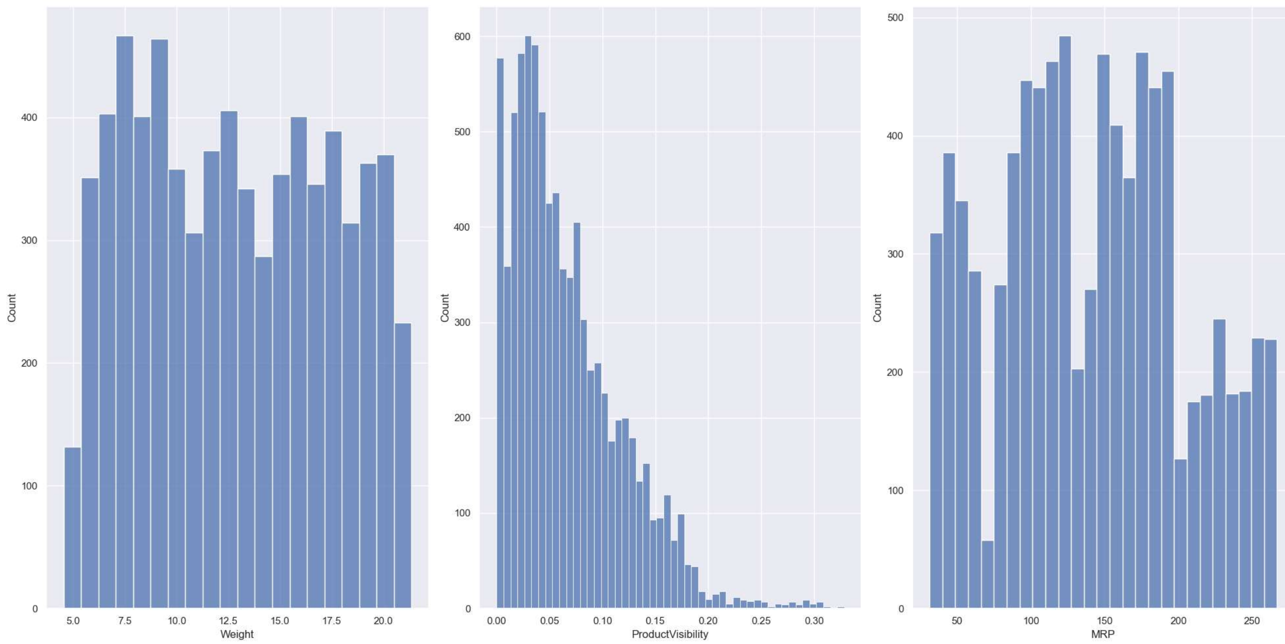
In [10]:
```
num_vars = ['Weight', 'ProductVisibility', 'MRP']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [11]:
```
num_vars = ['Weight', 'ProductVisibility', 'MRP']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()
plt.show()
```

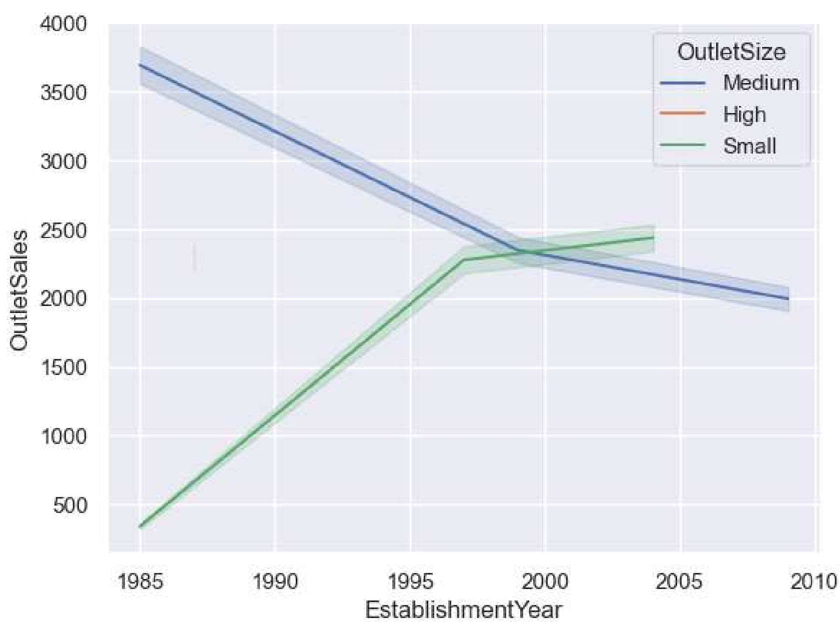


In [12]:
```
num_vars = ['Weight', 'ProductVisibility', 'MRP']

fig, axs = plt.subplots(nrows=1, ncols=3, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



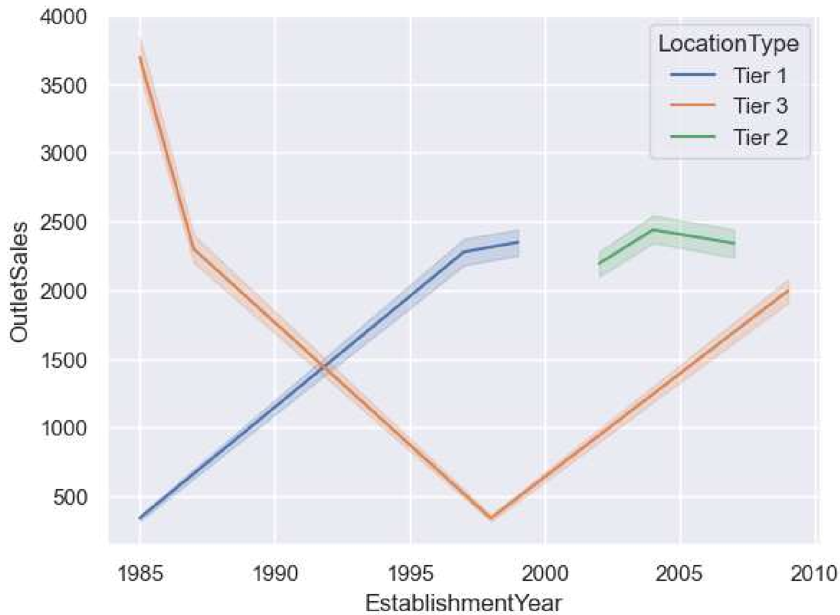
In [19]:
```
sns.lineplot(data=df, x="EstablishmentYear", y="OutletSales", hue="OutletSize")
```

Out[19]:
```
<AxesSubplot:xlabel='EstablishmentYear', ylabel='OutletSales'>
```



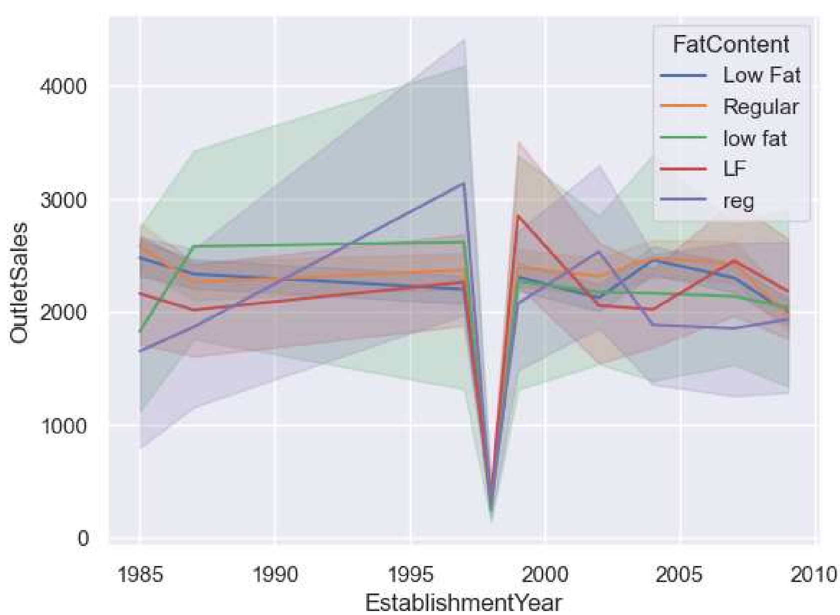
In [20]:
```
sns.lineplot(data=df, x="EstablishmentYear", y="OutletSales", hue="LocationType")
```

Out[20]: <AxesSubplot:xlabel='EstablishmentYear', ylabel='OutletSales'>



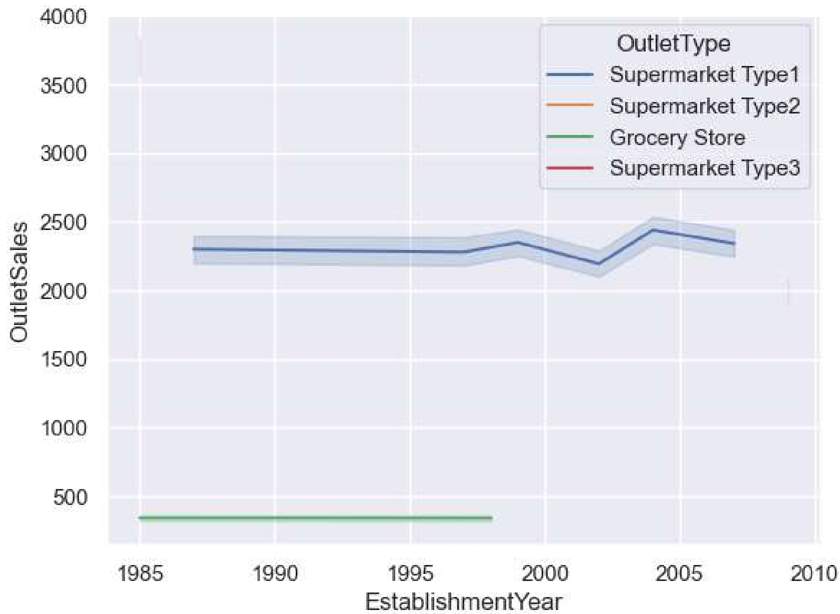
In [21]:
```
sns.lineplot(data=df, x="EstablishmentYear", y="OutletSales", hue="FatContent")
```

Out[21]: <AxesSubplot:xlabel='EstablishmentYear', ylabel='OutletSales'>



In [22]:
```
sns.lineplot(data=df, x="EstablishmentYear", y="OutletSales", hue="OutletType")
```

Out[22]:
```
<AxesSubplot:xlabel='EstablishmentYear', ylabel='OutletSales'>
```



Data Preprocessing Part 2

In [23]:
```
#Check the missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[23]:
```
OutletSize    28.276428
Weight        17.165317
dtype: float64
```

In [24]:
```
df.shape
```

Out[24]:
```
(8523, 11)
```

In [25]:
```
# fill the Weight size with mean
df['Weight'] = df['Weight'].fillna(df['Weight'].mean())
```

In [26]:
```
unique_sizes_train3 = df.groupby('OutletType')['OutletSize'].unique()
unique_sizes_train3
```

Out[26]:
```
OutletType
Grocery Store                          [nan, Small]
Supermarket Type1    [Medium, High, nan, Small]
Supermarket Type2                         [Medium]
Supermarket Type3                         [Medium]
Name: OutletSize, dtype: object
```

In [27]:
```
# fill null value in OutletSize where OutletType == Grocery Store with 'Small' value
df.loc[(df['OutletType'] == 'Grocery Store') & (df['OutletSize'].isna()), 'OutletSize'] = 'Small'
```

In [28]:
```
#Check the missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[28]:
```
OutletSize    21.764637
dtype: float64
```

In [29]:
```
#drop 'OutlietSize' null value row
df.dropna(subset=['OutletSize'], inplace=True)
df.shape
```

Out[29]:
```
(6668, 11)
```

In [30]:
```
df.head()
```

Out[30]:

	Weight	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType	OutletSales
0	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380
1	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228
2	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700
3	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	Small	Tier 3	Grocery Store	732.3800
4	8.93	Low Fat	0.000000	Household	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052

Label encoding for each object data type

In [31]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
FatContent: ['Low Fat' 'Regular' 'low fat' 'LF' 'reg']
ProductType: ['Dairy' 'Soft Drinks' 'Meat' 'Fruits and Vegetables' 'Household'
 'Baking Goods' 'Snack Foods' 'Breakfast' 'Health and Hygiene'
 'Hard Drinks' 'Frozen Foods' 'Canned' 'Starchy Foods' 'Others' 'Breads'
 'Seafood']
OutletID: ['OUT049' 'OUT018' 'OUT010' 'OUT013' 'OUT027' 'OUT046' 'OUT035' 'OUT019']
OutletSize: ['Medium' 'Small' 'High']
LocationType: ['Tier 1' 'Tier 3' 'Tier 2']
OutletType: ['Supermarket Type1' 'Supermarket Type2' 'Grocery Store'
 'Supermarket Type3']
```

In [32]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
FatContent: [1 2 3 0 4]
ProductType: [ 4 14 10  6  9  0 13  2  8  7  5  3 15 11  1 12]
OutletID: [7 2 0 1 4 6 5 3]
OutletSize: [1 2 0]
LocationType: [0 2 1]
OutletType: [1 2 0 3]
```

Remove Outlier using IQR

In [36]:
```
df.shape
```

Out[36]: (6668, 11)

In [38]:
```
# specify the columns to remove outliers from dataframe
column_names = ['ProductVisibility']

# remove outliers for each selected column using the IQR method
for column_name in column_names:
    Q1 = df[column_name].quantile(0.25)
    Q3 = df[column_name].quantile(0.75)
    IQR = Q3 - Q1
    df = df[~((df[column_name] < (Q1 - 1.5 * IQR)) | (df[column_name] > (Q3 + 1.5 * IQR)))]

df.head()
```

Out[38]:

	Weight	FatContent	ProductVisibility	ProductType	MRP	OutletID	EstablishmentYear	OutletSize	LocationType	OutletType	OutletSales
0	9.30	1	0.016047	4	249.8092	7	1999	1	0	1	3735.1380
1	5.92	2	0.019278	14	48.2692	2	2009	1	2	2	443.4228
2	17.50	1	0.016760	10	141.6180	7	1999	1	0	1	2097.2700
3	19.20	2	0.000000	6	182.0950	0	1998	2	2	0	732.3800
4	8.93	1	0.000000	9	53.8614	1	1987	0	2	1	994.7052

In [39]:
```
df.shape
```

Out[39]:
```
(6535, 11)
```

Heatmap Correlation

In [42]:
```
plt.figure(figsize=(15,12))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[42]: <AxesSubplot:>

Train Test Split

In [43]:
```
X = df.drop('OutletSales', axis=1)
y = df['OutletSales']
```

In [44]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0)
```

Decision Tree Regressor

In [45]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import load_boston


# Create a DecisionTreeRegressor object
dtree = DecisionTreeRegressor()

# Define the hyperparameters to tune and their values
param_grid = {
    'max_depth': [2, 4, 6, 8],
    'min_samples_split': [2, 4, 6, 8],
    'min_samples_leaf': [1, 2, 3, 4],
    'max_features': ['auto', 'sqrt', 'log2']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(dtree, param_grid, cv=5, scoring='neg_mean_squared_error')

# Fit the GridSearchCV object to the data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 6, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 2}
```

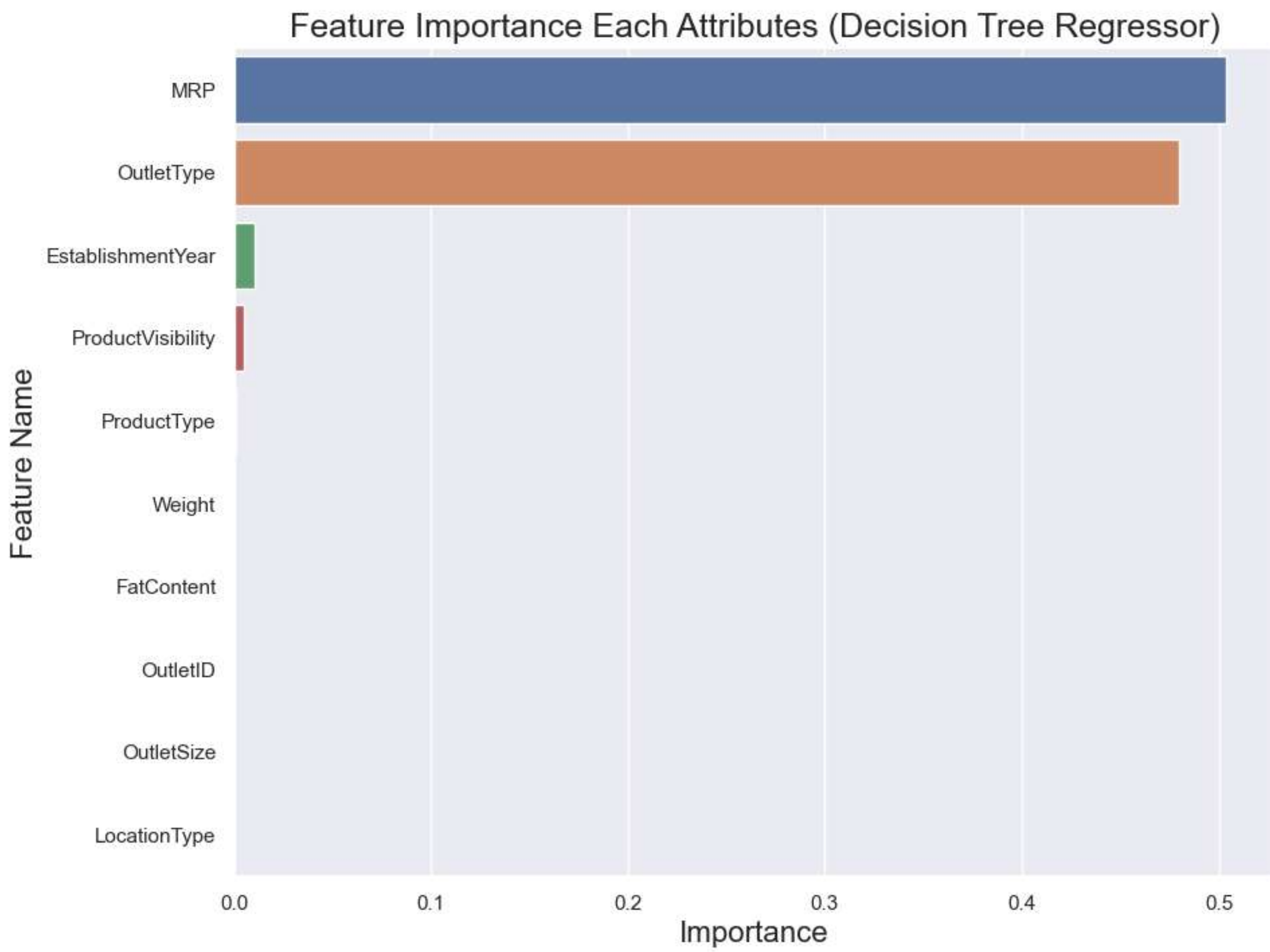
In [46]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=6, max_features='auto', min_samples_leaf=4, min_samples_split=
dtree.fit(X_train, y_train)
```

Out[46]:
```
DecisionTreeRegressor(max_depth=6, max_features='auto', min_samples_leaf=4,
                      random_state=0)
```

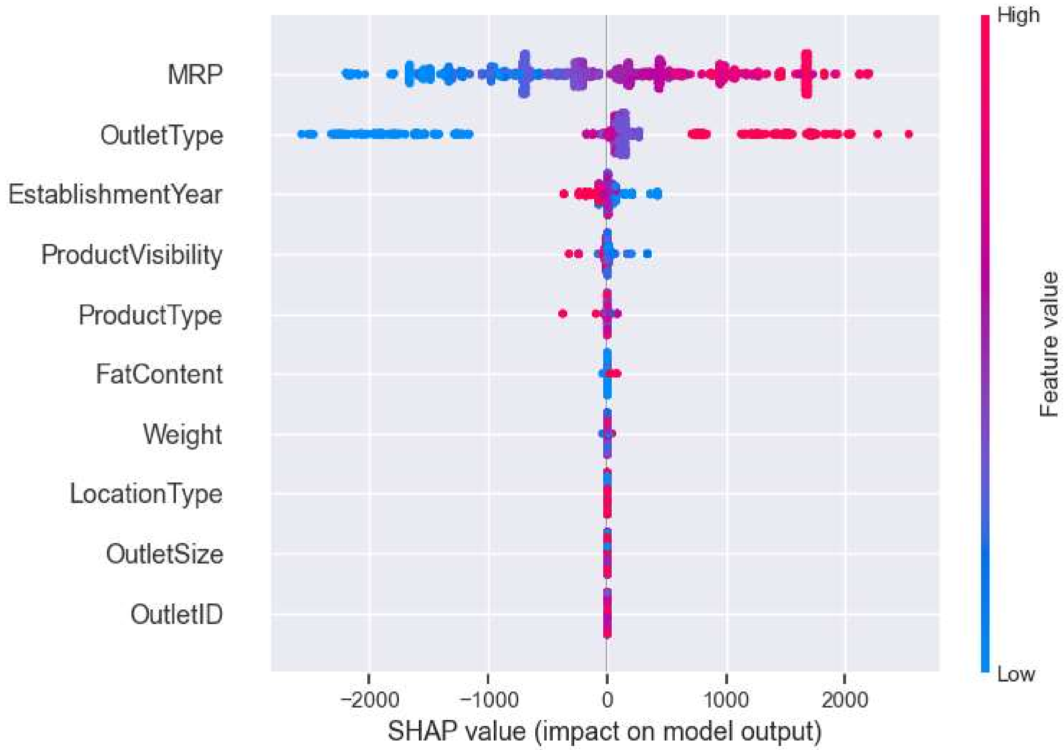
In [47]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 764.98722800516
MAPE is 0.5784403446838259
MSE is 1245334.0275746458
R2 score is 0.6233611975537985
RMSE score is 1115.945351518006
```

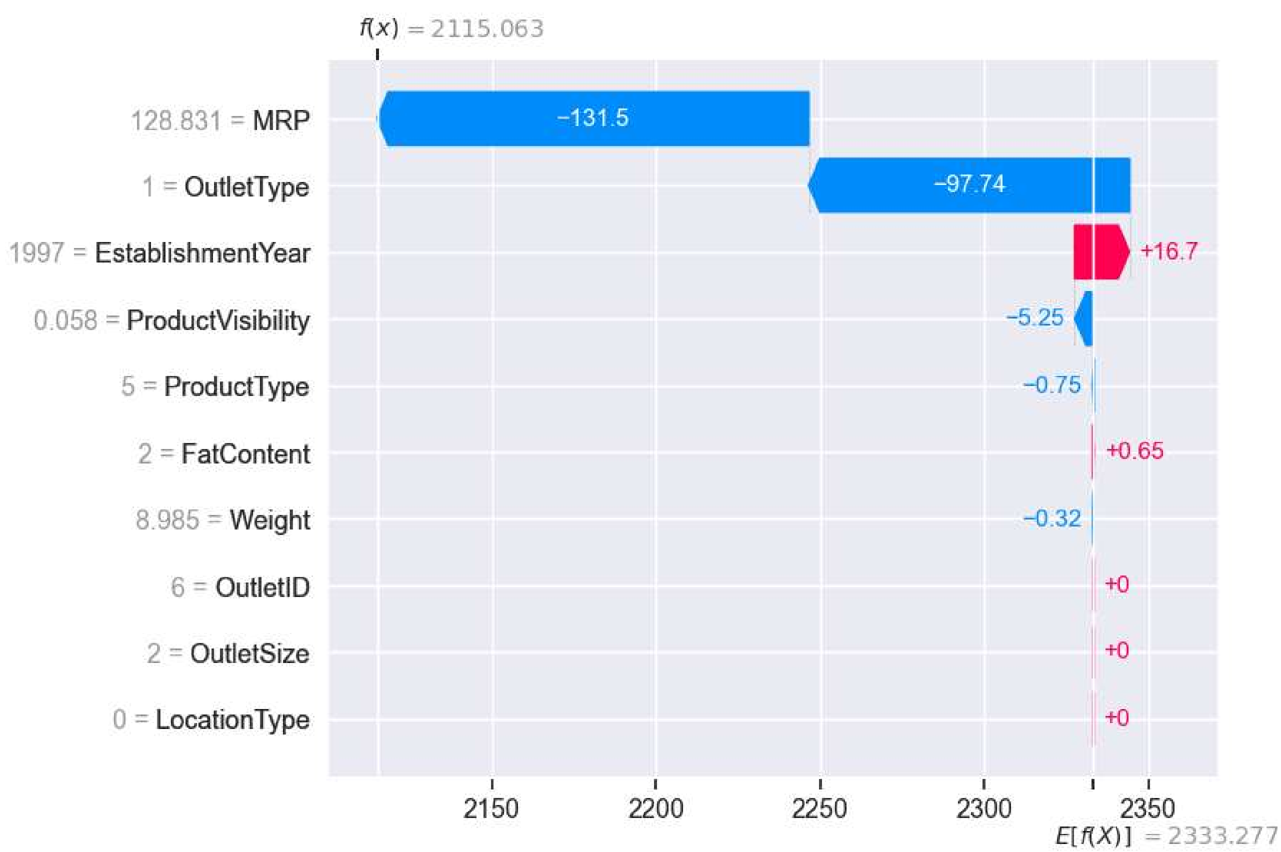
In [48]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Decision Tree Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [49]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [50]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



Random Forest Regressor

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

# Create a Random Forest Regressor object
rf = RandomForestRegressor()

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7, 9],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': ['auto', 'sqrt']
}

# Create a GridSearchCV object
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

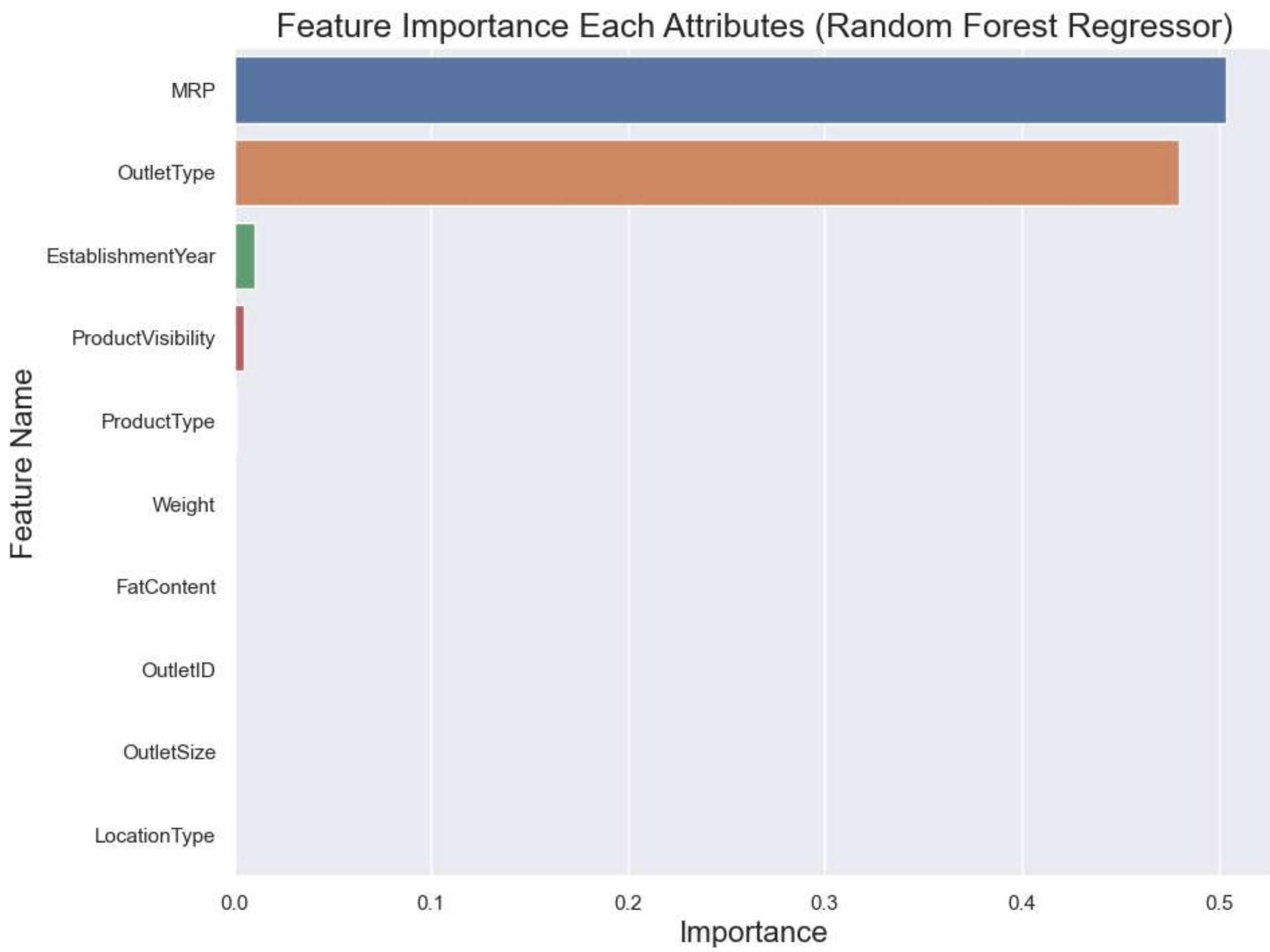
In [52]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=5, min_samples_split=2, min_samples_leaf=4,
                           max_features='auto')
rf.fit(X_train, y_train)
```

Out[52]:
```
RandomForestRegressor(max_depth=5, min_samples_leaf=4, random_state=0)
```

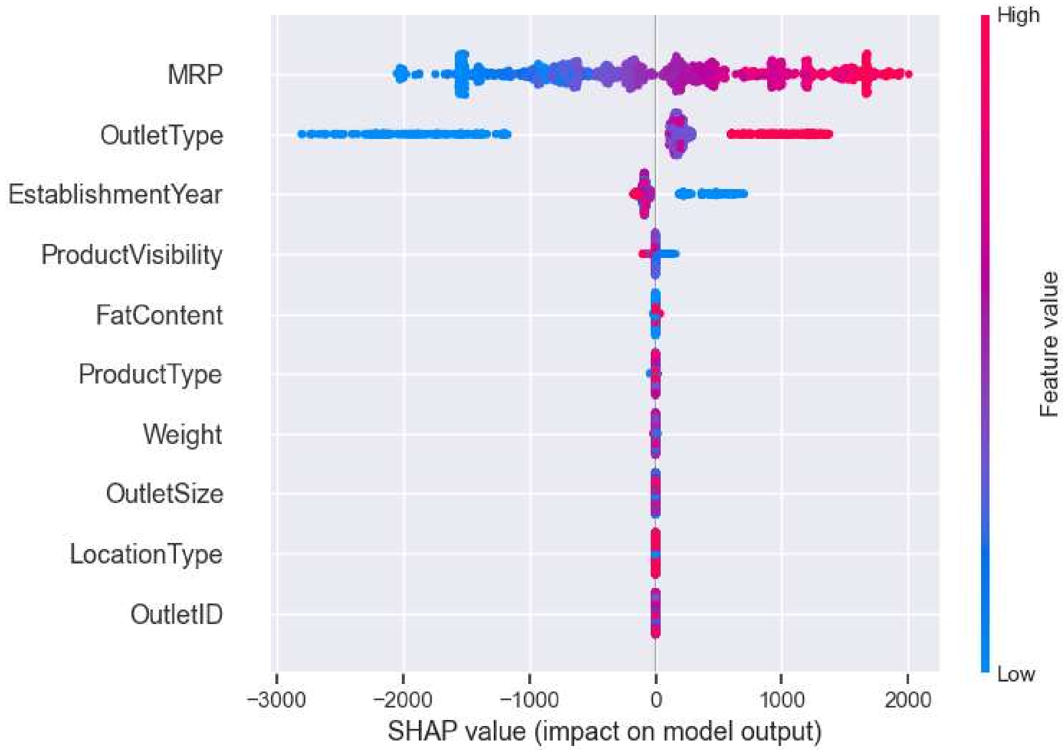
In [53]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 753.013521857796
MAPE is 0.579744877093664
MSE is 1196954.1974070177
R2 score is 0.6379931925795618
RMSE score is 1094.0540194190676
```

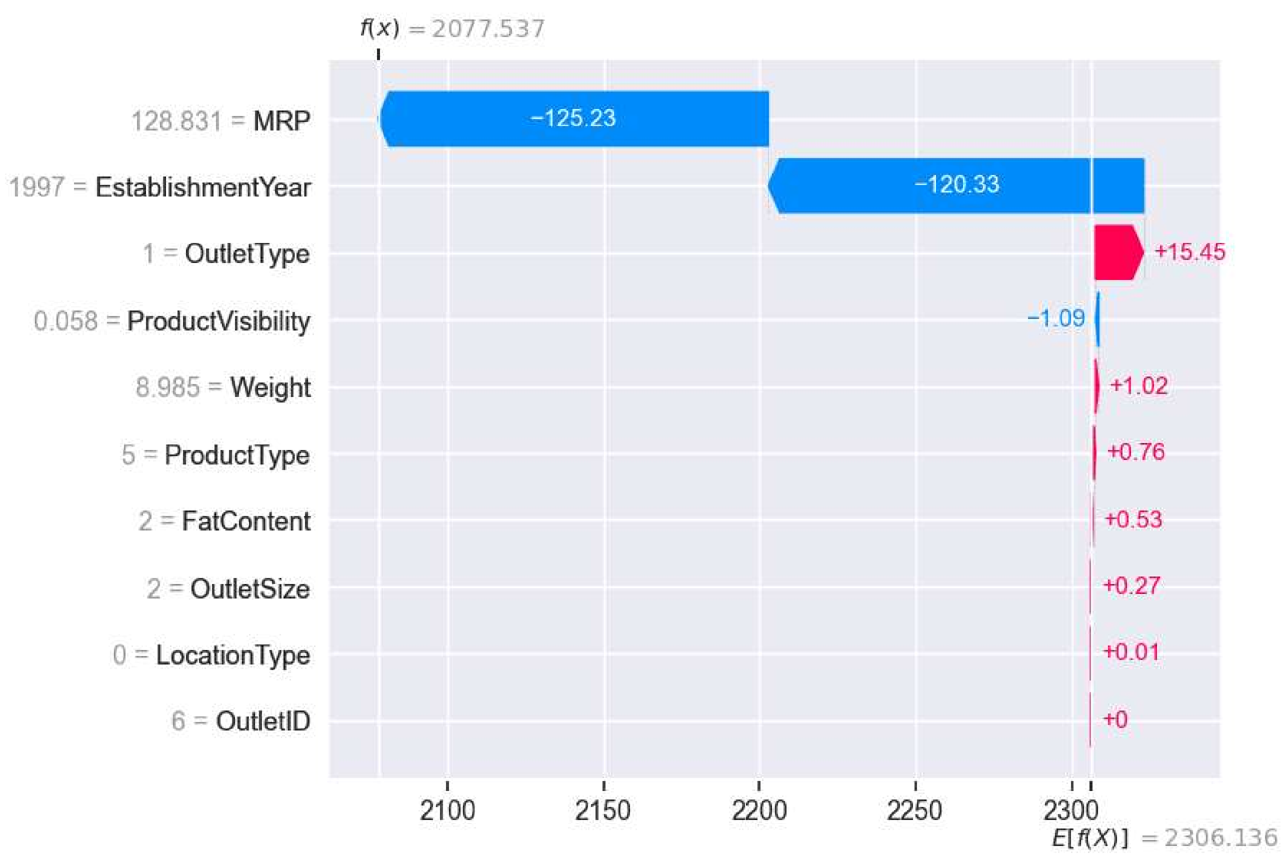
In [54]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Feature Importance Each Attributes (Random Forest Regressor)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [55]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [56]:
```
explainer = shap.Explainer(rf, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```



XGBoost Regressor

In [57]:
```
from xgboost import XGBRegressor
from sklearn.model_selection import GridSearchCV

# Create an XGBoost Regressor object
xgb = XGBRegressor()

# Define the hyperparameter grid
param_grid = {
    'max_depth': [3, 5, 7, 9],
    'min_child_weight': [1, 3, 5],
    'learning_rate': [0.1, 0.01, 0.001],
    'gamma': [0, 1, 5]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(xgb, param_grid, cv=5, scoring='r2')

# Fit the GridSearchCV object to the training data
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print("Best hyperparameters: ", grid_search.best_params_)
```

```
Best hyperparameters:  {'gamma': 0, 'learning_rate': 0.1, 'max_depth': 3, 'min_child_weight': 1}
```

In [58]:
```
from sklearn.ensemble import RandomForestRegressor
xgb = XGBRegressor(max_depth=3, min_child_weight=1, learning_rate=0.1, gamma=0)
xgb.fit(X_train, y_train)
```

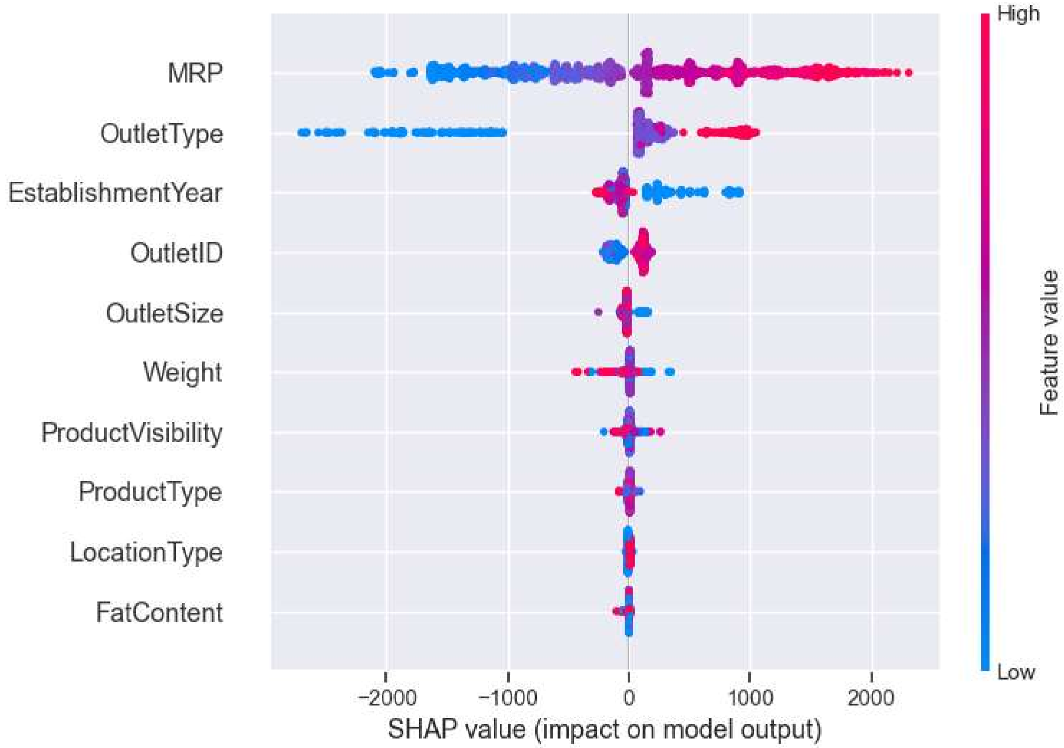
Out[58]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=0, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=3, max_leaves=None,
             min_child_weight=1, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=None, ...)
```

In [59]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = xgb.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, y_pred)
rmse = math.sqrt(mse)

print('MAE is {}'.format(mae))
print('MAPE is {}'.format(mape))
print('MSE is {}'.format(mse))
print('R2 score is {}'.format(r2))
print('RMSE score is {}'.format(rmse))
```

```
MAE is 754.4837491406836
MAPE is 0.5890650529183769
MSE is 1196676.6797329972
R2 score is 0.6380771250202455
RMSE score is 1093.9271820980578
```

In [60]:
```
import shap
explainer = shap.TreeExplainer(xgb)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
ntree_limit is deprecated, use `iteration_range` or model slicing instead.
```



In [61]:
```
explainer = shap.Explainer(xgb, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```