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]:
```
df = pd.read_csv('TRAIN.csv')
df.head()
```

Out[2]:

	ID	Store_id	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
0	T1000001	1	S1	L3	R1	2018-01-01	1	Yes	9	7011.84
1	T1000002	253	S4	L2	R1	2018-01-01	1	Yes	60	51789.12
2	T1000003	252	S3	L2	R1	2018-01-01	1	Yes	42	36868.20
3	T1000004	251	S2	L3	R1	2018-01-01	1	Yes	23	19715.16
4	T1000005	250	S2	L3	R4	2018-01-01	1	Yes	62	45614.52

Data Preprocessing Part 1

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

Out[3]:
```
ID               188340
Store_Type            4
Location_Type         5
Region_Code           4
Date                516
Discount              2
dtype: int64
```

In [4]:
```
# Number of Store_id unique value
df.Store_id.nunique()
```

Out[4]: 365

In [5]:
```
# Drop ID and Store ID Column because its unnecesary
df.drop(columns=['ID','Store_id'], inplace=True)
df.head()
```

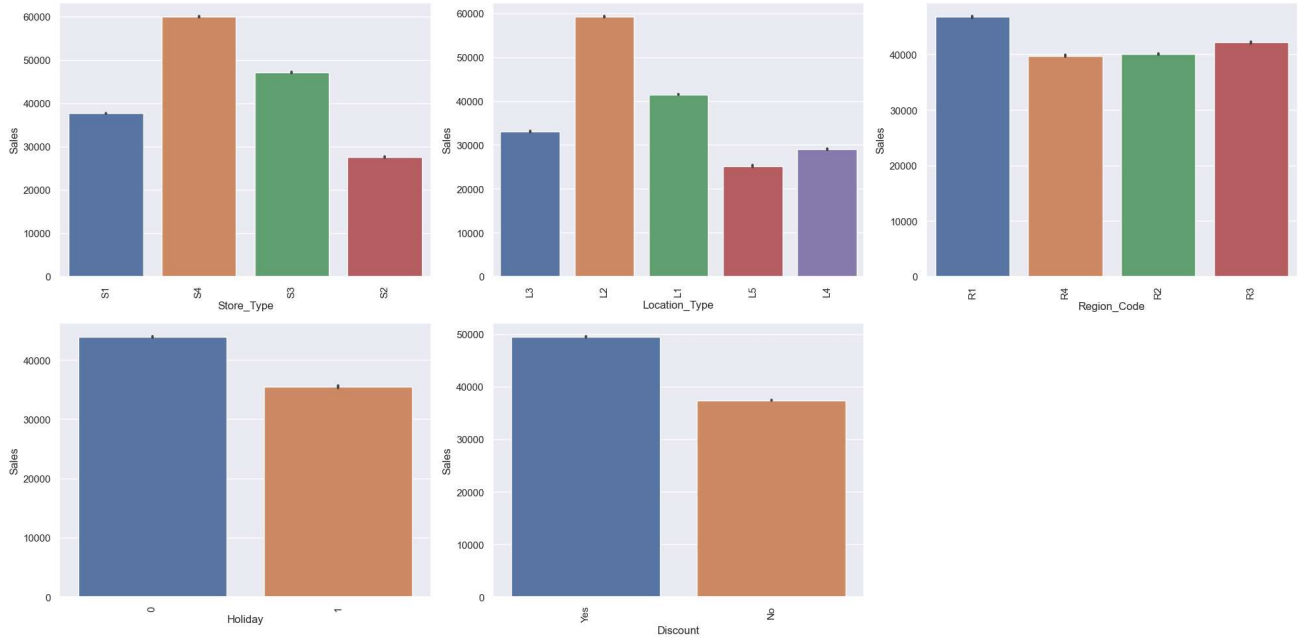
Out[5]:

	Store_Type	Location_Type	Region_Code	Date	Holiday	Discount	#Order	Sales
0	S1	L3	R1	2018-01-01	1	Yes	9	7011.84
1	S4	L2	R1	2018-01-01	1	Yes	60	51789.12
2	S3	L2	R1	2018-01-01	1	Yes	42	36868.20
3	S2	L3	R1	2018-01-01	1	Yes	23	19715.16
4	S2	L3	R4	2018-01-01	1	Yes	62	45614.52

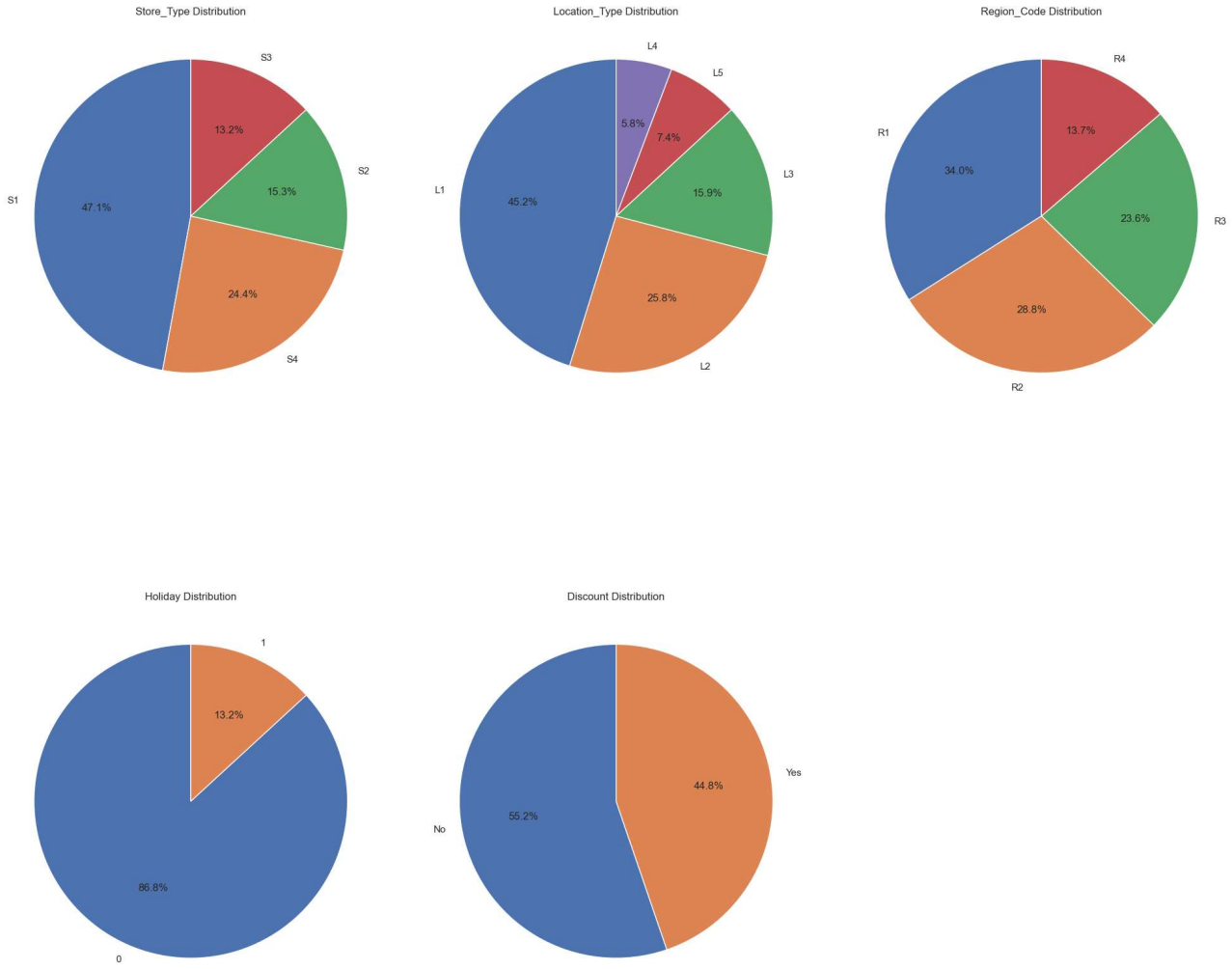
In [6]:
```
# Convert 'Date' column to datetime
df['Date'] = pd.to_datetime(df['Date'])
```

Exploratory Data Analysis

In [7]:
```
# list of categorical variables to plot
cat_vars = ['Store_Type', 'Location_Type', 'Region_Code', 'Holiday',
            'Discount']

# 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='Sales', data=df, ax=axs[i], estimator=np.mean)
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# remove the sixth subplot
fig.delaxes(axs[5])

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```

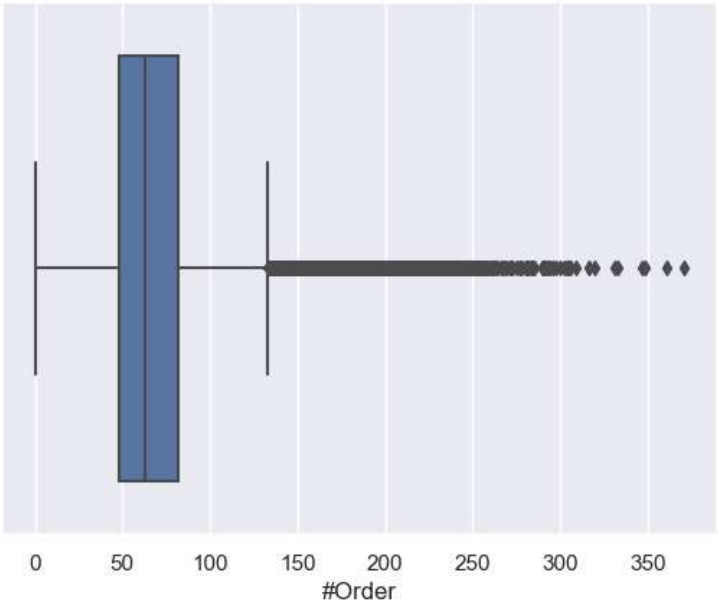


In [8]:
```
cat_vars = ['Store_Type', 'Location_Type', 'Region_Code', 'Holiday',
            'Discount']

# create a figure and axes
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(20, 20))

# 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()

# remove sixt plot
fig.delaxes(axs[1][2])

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



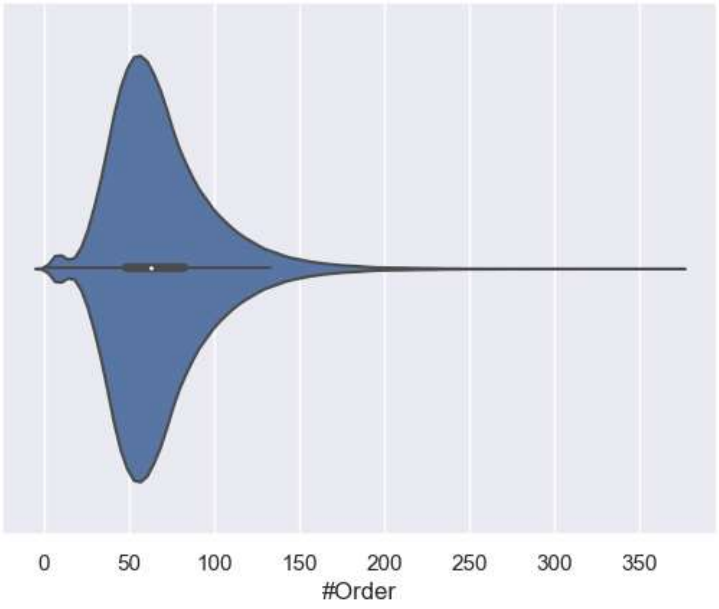
In [9]:
```
sns.boxplot(x='#Order', data=df)
```

Out[9]:
```
<AxesSubplot:xlabel='#Order'>
```

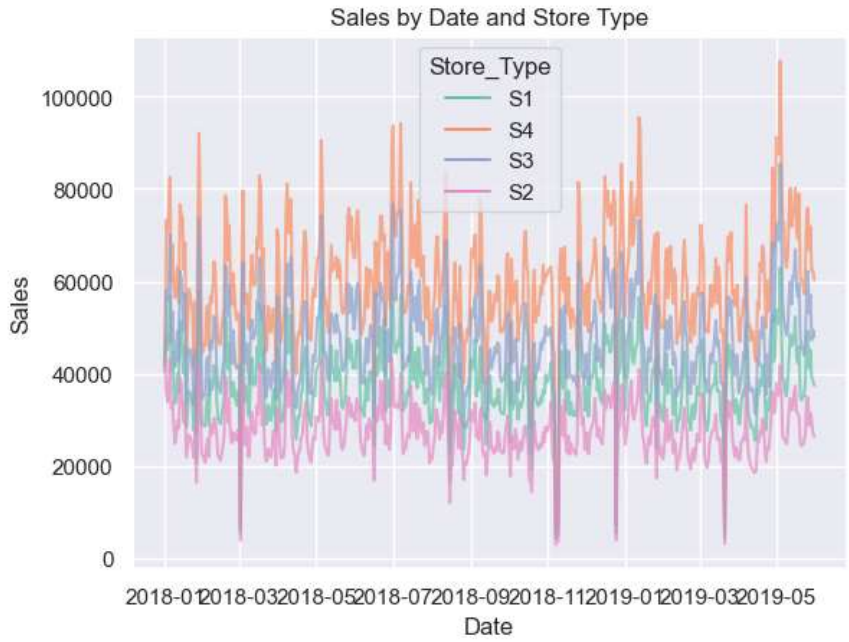


In [10]:
```
sns.violinplot(x='#Order', data=df)
```

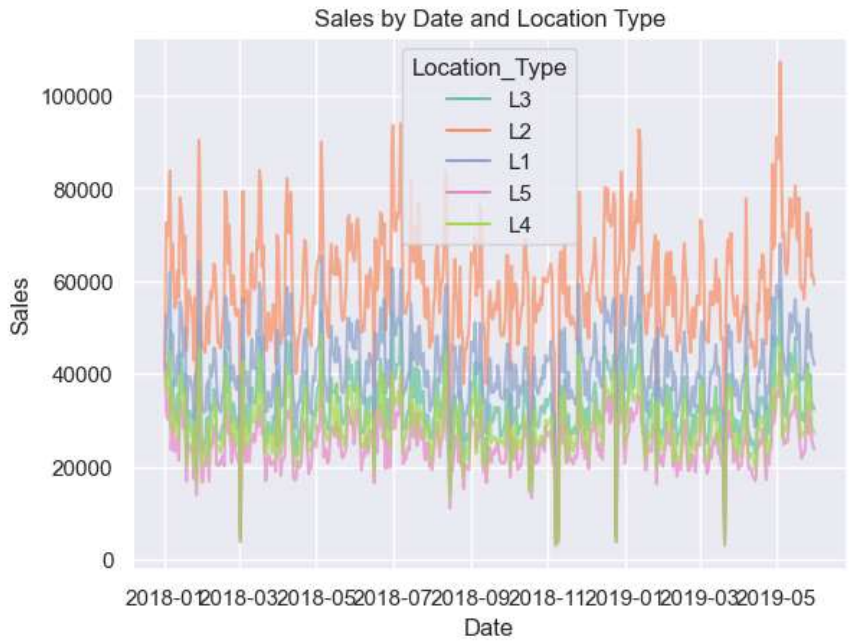
Out[10]:
```
<AxesSubplot:xlabel='#Order'>
```



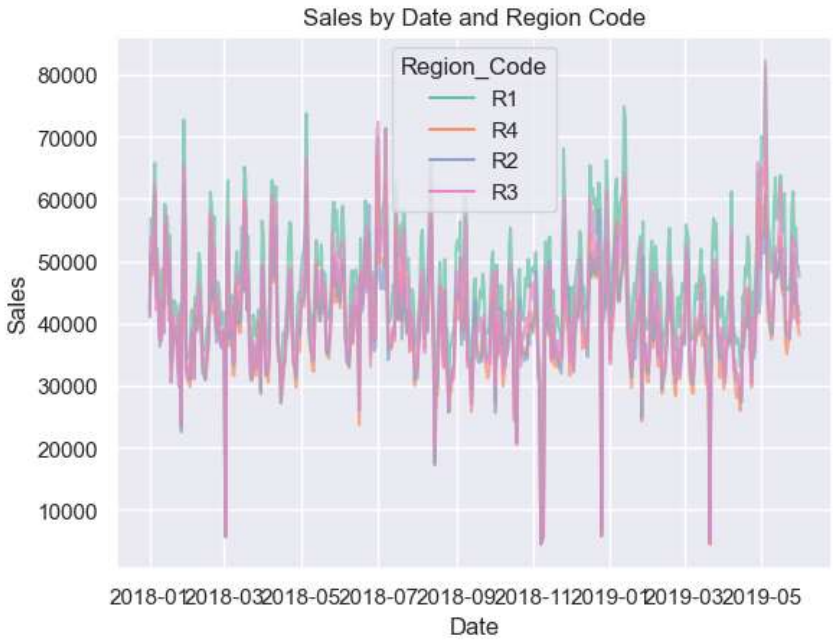
In [11]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Date', y='Sales', hue='Store_Type', data=df, ci=None, estimator='mean', alpha=0.7)

plt.title("Sales by Date and Store Type")
plt.xlabel("Date")
plt.ylabel("Sales")

plt.show()
```



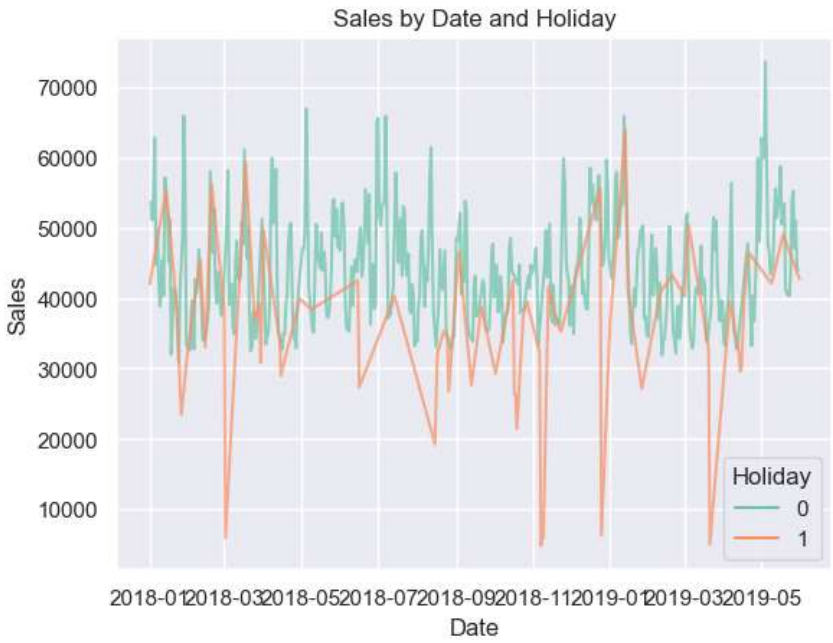
In [12]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Date', y='Sales', hue='Location_Type', data=df, ci=None, estimator='mean', alpha=0.7)

plt.title("Sales by Date and Location Type")
plt.xlabel("Date")
plt.ylabel("Sales")

plt.show()
```



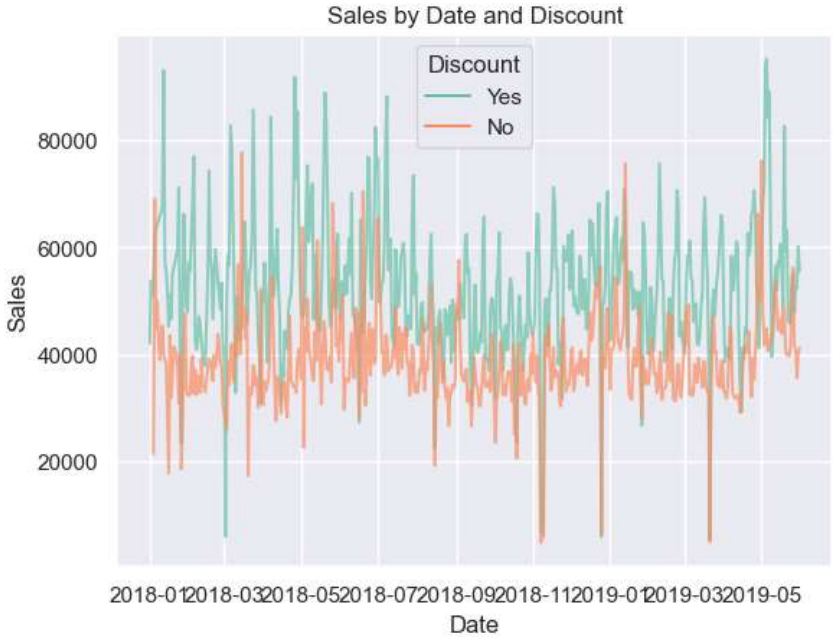
In [13]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Date', y='Sales', hue='Region_Code', data=df, ci=None, estimator='mean', alpha=0.7)

plt.title("Sales by Date and Region Code")
plt.xlabel("Date")
plt.ylabel("Sales")

plt.show()
```



In [14]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Date', y='Sales', hue='Holiday', data=df, ci=None, estimator='mean', alpha=0.7)

plt.title("Sales by Date and Holiday")
plt.xlabel("Date")
plt.ylabel("Sales")

plt.show()
```

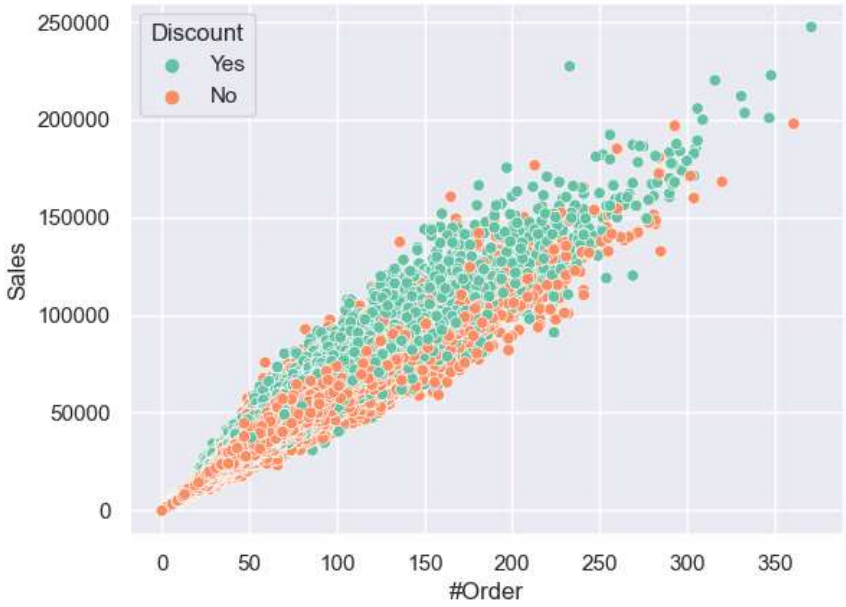


In [15]:
```
sns.set_style("darkgrid")
sns.set_palette("Set2")

sns.lineplot(x='Date', y='Sales', hue='Discount', data=df, ci=None, estimator='mean', alpha=0.7)

plt.title("Sales by Date and Discount")
plt.xlabel("Date")
plt.ylabel("Sales")

plt.show()
```



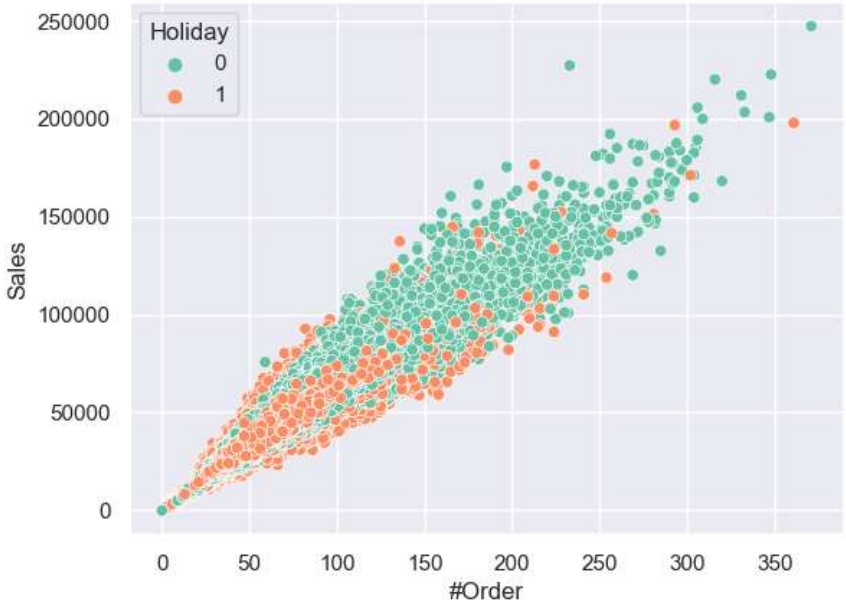
In [16]:
```
sns.scatterplot(x='#Order', y='Sales', hue='Discount', data=df)
```

Out[16]:
```
<AxesSubplot:xlabel='#Order', ylabel='Sales'>
```



In [17]:
```
sns.scatterplot(x='#Order', y='Sales', hue='Holiday', data=df)
```

Out[17]:
```
<AxesSubplot:xlabel='#Order', ylabel='Sales'>
```



Data Preprocessing Part 2

In [18]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[18]:
```
Series([], dtype: float64)
```

In [19]:
```
df.shape
```

Out[19]:
```
(188340, 8)
```

In [20]:
```
df.drop(columns='Date', inplace=True)
df.head()
```

Out[20]:

	Store_Type	Location_Type	Region_Code	Holiday	Discount	#Order	Sales
0	S1	L3	R1	1	Yes	9	7011.84
1	S4	L2	R1	1	Yes	60	51789.12
2	S3	L2	R1	1	Yes	42	36868.20
3	S2	L3	R1	1	Yes	23	19715.16
4	S2	L3	R4	1	Yes	62	45614.52

Label Encoding for Object datatype

In [21]:
```
# 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()}")
```

```
Store_Type: ['S1' 'S4' 'S3' 'S2']
Location_Type: ['L3' 'L2' 'L1' 'L5' 'L4']
Region_Code: ['R1' 'R4' 'R2' 'R3']
Discount: ['Yes' 'No']
```

In [22]:
```
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()}")
```

```
Store_Type: [0 3 2 1]
Location_Type: [2 1 0 4 3]
Region_Code: [0 3 1 2]
Discount: [1 0]
```

Correlation Heatmap

In [23]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[23]: <AxesSubplot:>

Train Test Split

In [24]:
```
from sklearn.model_selection import train_test_split

# Perform train-test split
X_train, X_test, y_train, y_test = train_test_split(df.drop('Sales', axis=1), df['Sales'], test_size=0.2, random_state
```

Remove Outlier Using IQR

In [25]:
```
# Concatenate X_train and y_train for outlier removal
train_df = pd.concat([X_train, y_train], axis=1)

# Calculate the IQR values for each column
Q1 = train_df.quantile(0.25)
Q3 = train_df.quantile(0.75)
IQR = Q3 - Q1

# Remove outliers from X_train
train_df = train_df[~((train_df < (Q1 - 1.5 * IQR)) | (train_df > (Q3 + 1.5 * IQR))).any(axis=1)]

# Separate X_train and y_train after outlier removal
X_train = train_df.drop('Sales', axis=1)
y_train = train_df['Sales']
```

Decision Tree Regressor

In [26]:
```
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': 8, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 2}
```

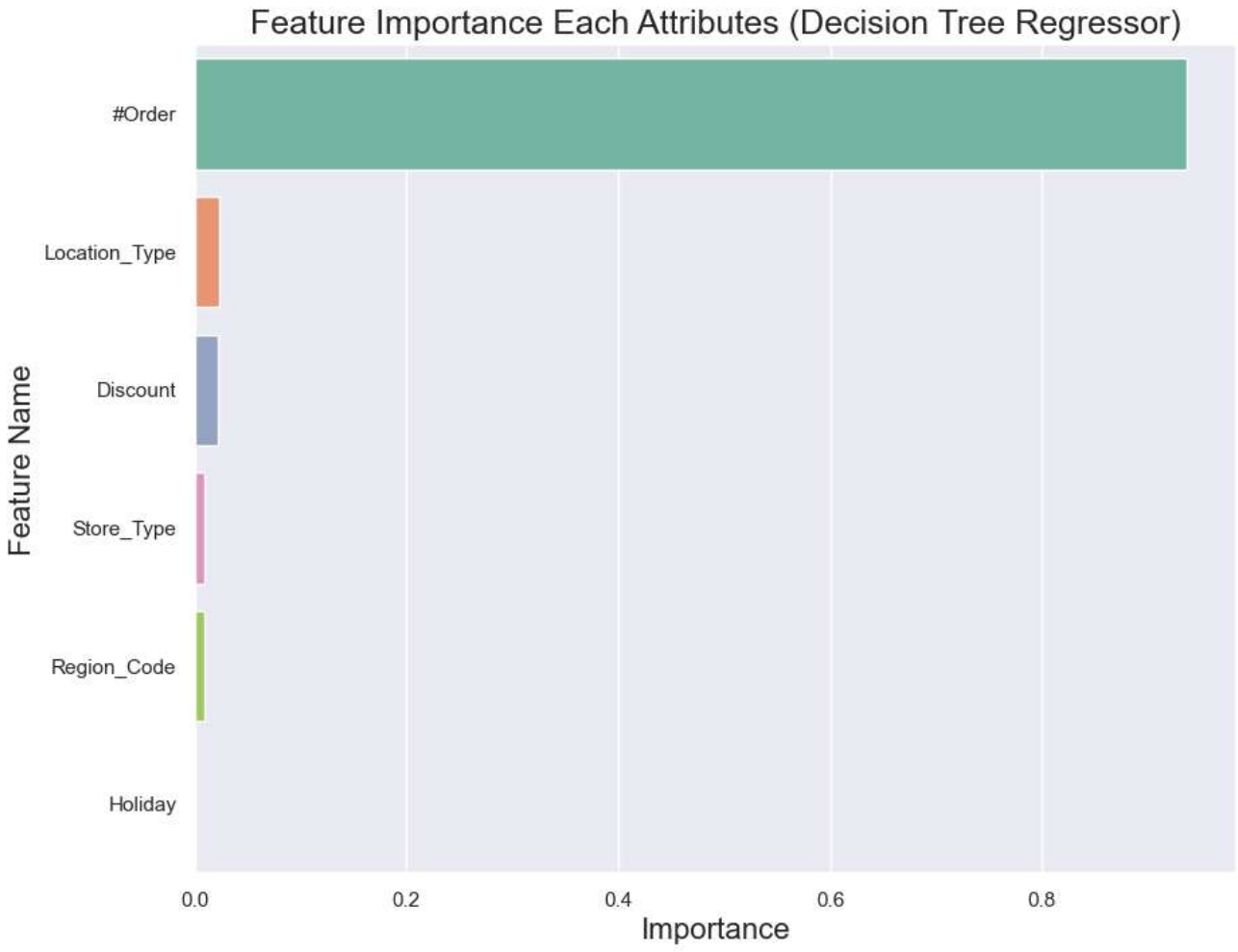
In [27]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=8, max_features='auto', min_samples_leaf=4, min_samples_split=
dtree.fit(X_train, y_train)
```

Out[27]:
```
DecisionTreeRegressor(max_depth=8, max_features='auto', min_samples_leaf=4,
                      random_state=0)
```

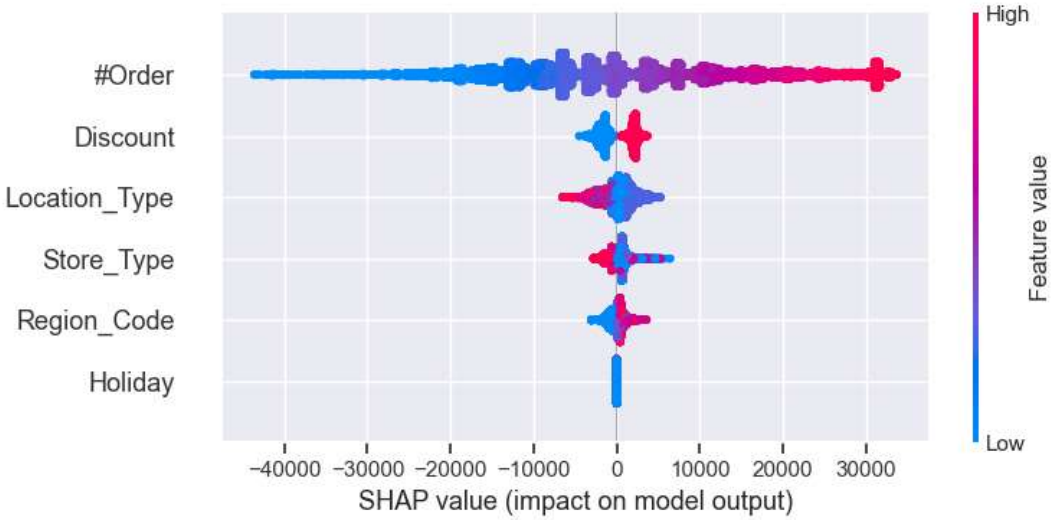
In [28]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = 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 4212.6722261862915
MAPE is 94620068628571.05
MSE is 50588389.954528056
R2 score is 0.8503541542191435
RMSE score is 7112.551578338681
```

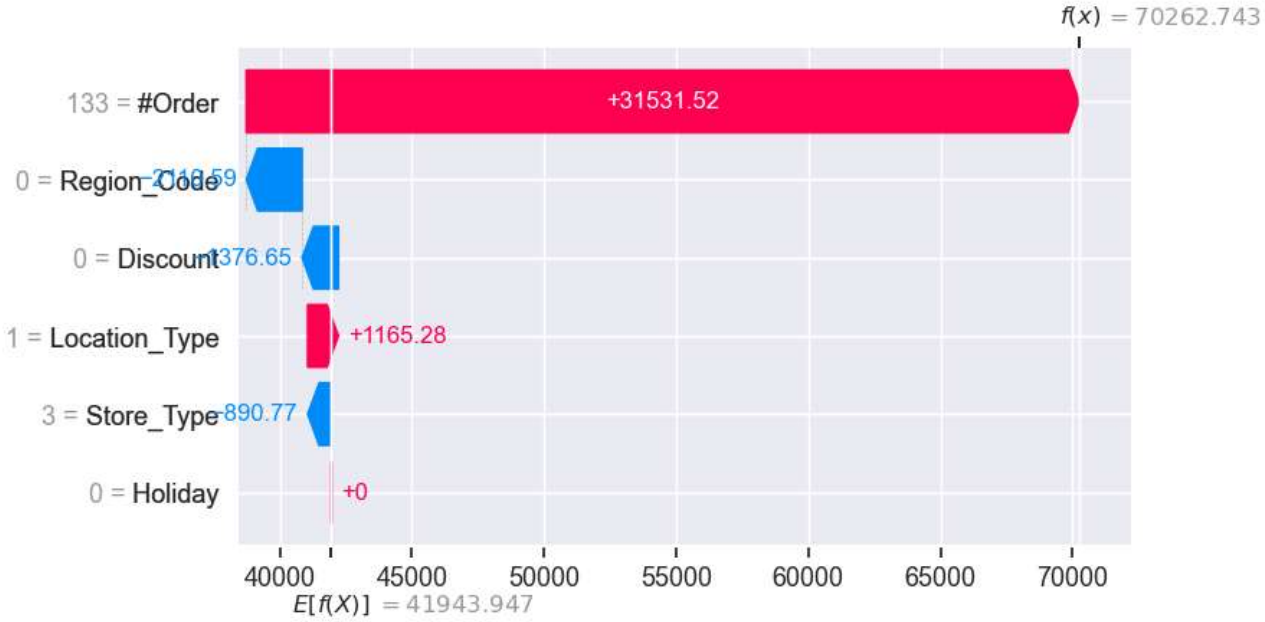
In [29]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 [30]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [32]:
```
explainer = shap.Explainer(dtree, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```



Random Forest Regressor

In [33]:
```
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_)
```

```
Best hyperparameters:  {'max_depth': 9, 'max_features': 'auto', 'min_samples_leaf': 2, 'min_samples_split': 5}
```

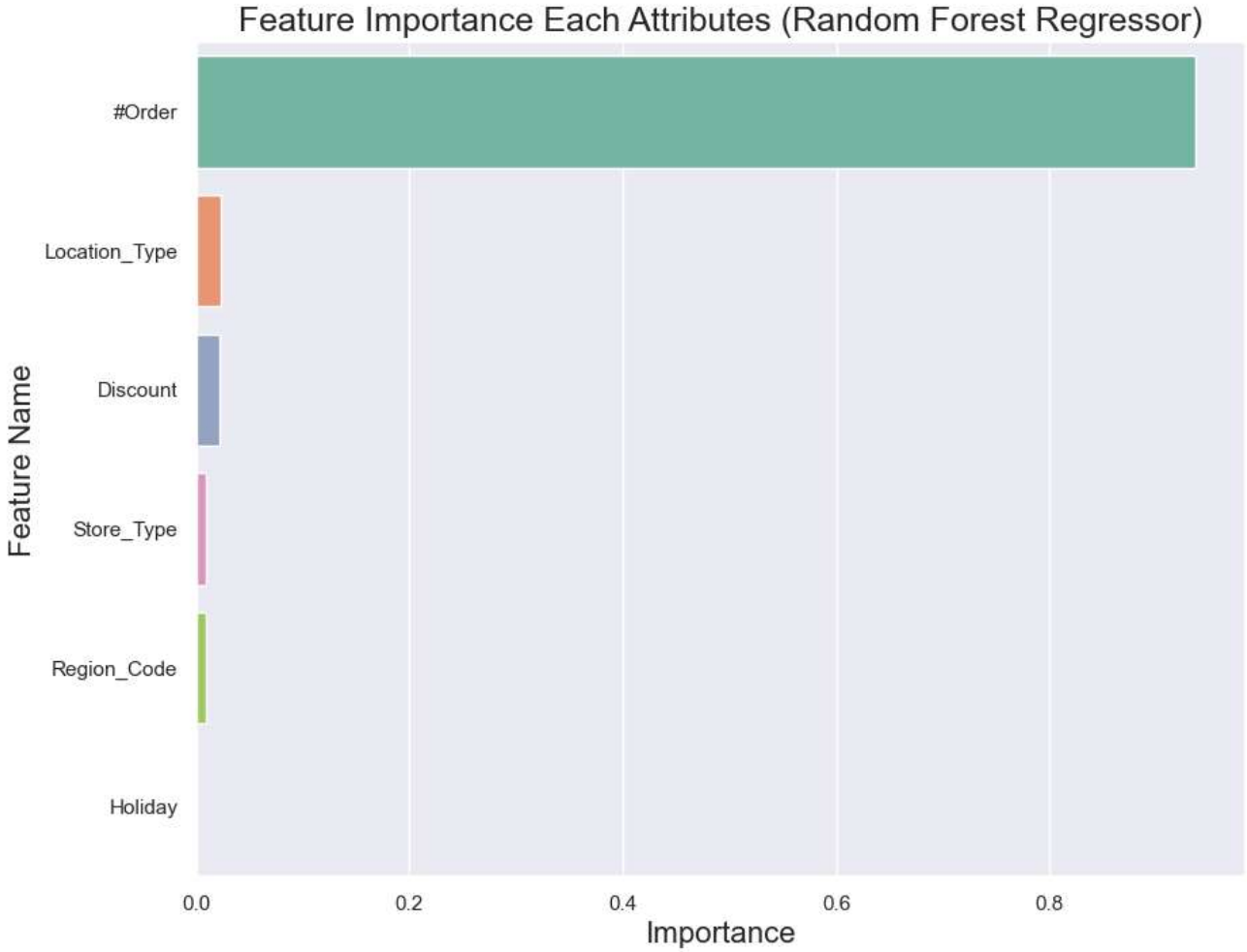
In [34]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=9, min_samples_split=5, min_samples_leaf=2,
                           max_features='auto')
rf.fit(X_train, y_train)
```

Out[34]:
```
RandomForestRegressor(max_depth=9, min_samples_leaf=2, min_samples_split=5,
                      random_state=0)
```

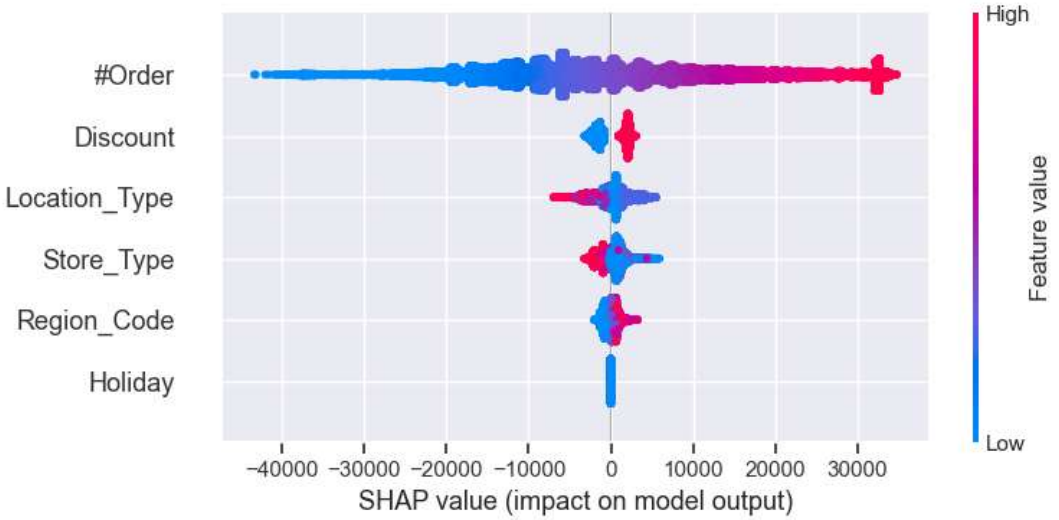
In [35]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, 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 3952.7234280476564
MAPE is 13877610065523.824
MSE is 46179049.597152114
R2 score is 0.8633974526460808
RMSE score is 6795.516874907465
```

In [36]:
```
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 [37]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [38]:
```
explainer = shap.Explainer(rf, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```

```
100%|==================| 37522/37668 [04:18<00:01]
```