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

In [2]:
```
df = pd.read_csv('House_Rent_Dataset.csv')
df.head()
```

Out[2]:

	Posted On	BHK	Rent	Size	Floor	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred
0	2022-05-18	2	10000	1100	Ground out of 2	Super Area	Bandel	Kolkata	Unfurnished	Bachelors/Family
1	2022-05-13	2	20000	800	1 out of 3	Super Area	Phool Bagan, Kankurgachi	Kolkata	Semi-Furnished	Bachelors/Family
2	2022-05-16	2	17000	1000	1 out of 3	Super Area	Salt Lake City Sector 2	Kolkata	Semi-Furnished	Bachelors/Family
3	2022-07-04	2	10000	800	1 out of 2	Super Area	Dumdum Park	Kolkata	Unfurnished	Bachelors/Family
4	2022-05-09	2	7500	850	1 out of 2	Carpet Area	South Dum Dum	Kolkata	Unfurnished	Bachelors

Data Preprocessing Part 1

In [3]:
```
# Drop 'Posted On' column
df.drop(columns = 'Posted On', inplace=True)
df.head()
```

Out[3]:

	BHK	Rent	Size	Floor	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred	Bathroo
0	2	10000	1100	Ground out of 2	Super Area	Bandel	Kolkata	Unfurnished	Bachelors/Family	
1	2	20000	800	1 out of 3	Super Area	Phool Bagan, Kankurgachi	Kolkata	Semi-Furnished	Bachelors/Family	
2	2	17000	1000	1 out of 3	Super Area	Salt Lake City Sector 2	Kolkata	Semi-Furnished	Bachelors/Family	
3	2	10000	800	1 out of 2	Super Area	Dumdum Park	Kolkata	Unfurnished	Bachelors/Family	
4	2	7500	850	1 out of 2	Carpet Area	South Dum Dum	Kolkata	Unfurnished	Bachelors	

In [4]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[4]:
```
Floor                480
Area Type              3
Area Locality       2235
City                   6
Furnishing Status      3
Tenant Preferred       3
Point of Contact       3
dtype: int64
```

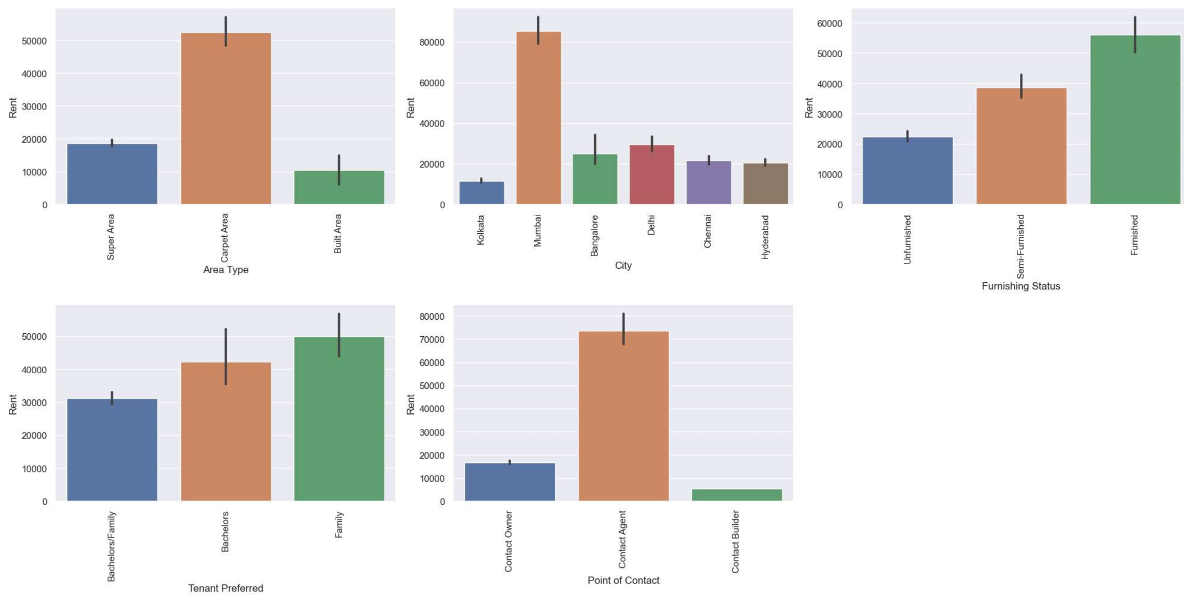
In [5]:
```
# Drop 'Floor' and 'Area Locality' column because of the amount of unique value
df.drop(columns= ['Floor', 'Area Locality'], inplace=True)
df.head()
```

Out[5]:

	BHK	Rent	Size	Area Type	City	Furnishing Status	Tenant Preferred	Bathroom	Point of Contact
0	2	10000	1100	Super Area	Kolkata	Unfurnished	Bachelors/Family	2	Contact Owner
1	2	20000	800	Super Area	Kolkata	Semi-Furnished	Bachelors/Family	1	Contact Owner
2	2	17000	1000	Super Area	Kolkata	Semi-Furnished	Bachelors/Family	1	Contact Owner
3	2	10000	800	Super Area	Kolkata	Unfurnished	Bachelors/Family	1	Contact Owner
4	2	7500	850	Carpet Area	Kolkata	Unfurnished	Bachelors	1	Contact Owner

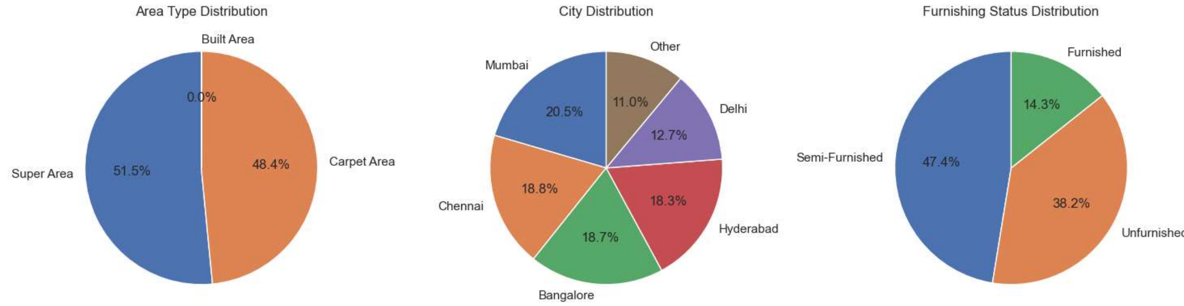
Exploratory Data Analysis

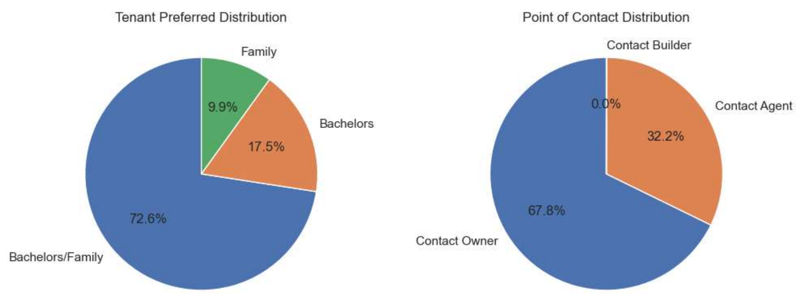
In [7]:
```
# list of categorical variables to plot
cat_vars = ['Area Type', 'City', 'Furnishing Status', 'Tenant Preferred', 'Poi

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

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

# adjust spacing between subplots
fig.tight_layout()

# remove the 6th subplot
fig.delaxes(axs[5])

# show plot
plt.show()
```



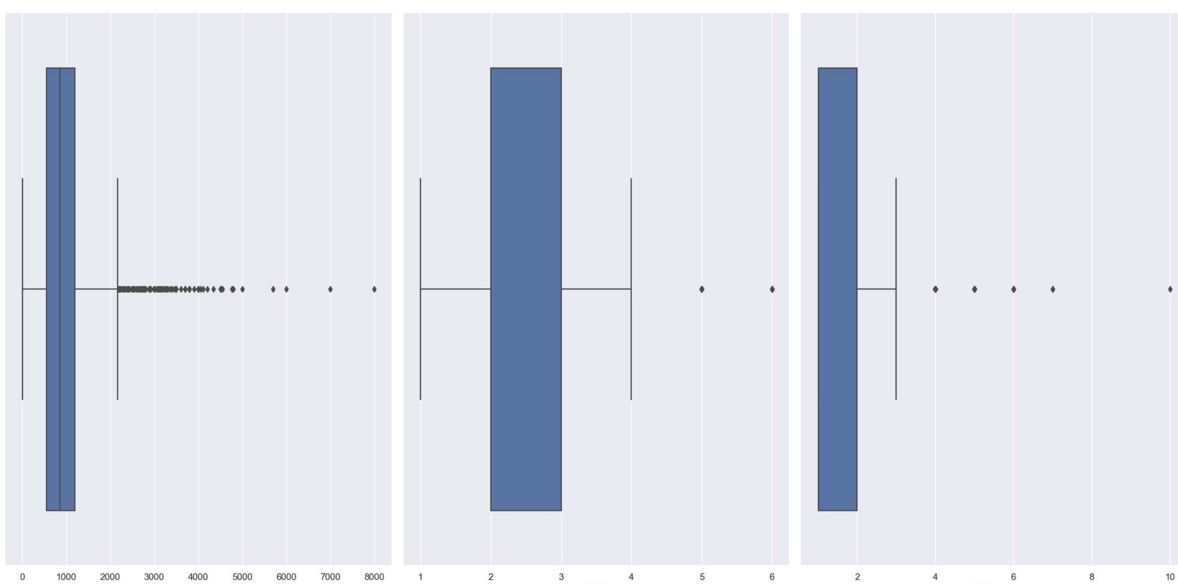
In [9]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

cat_vars = ['Area Type', 'City', 'Furnishing Status', 'Tenant Preferred', 'Poi

# 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()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), in
            cat_counts = cat_counts_top.append(cat_counts_other)

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

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

# Remove the 6th subplot
fig.delaxes(axs[1, 2])

# Adjust spacing between subplots
fig.tight_layout()

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

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_7272\1820397353.py:19: FutureWa
rning: The series.append method is deprecated and will be removed from pandas
in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```

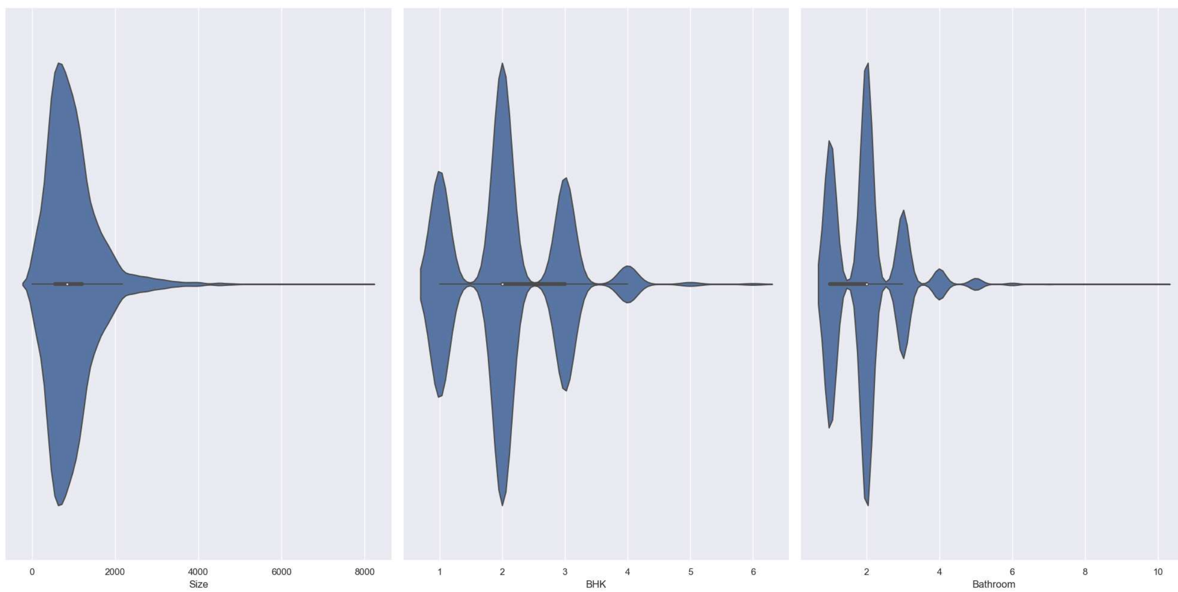




In [10]:
```python
num_vars = ['Size', 'BHK', 'Bathroom']

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 = ['Size', 'BHK', 'Bathroom']

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()
```



Data Preprocessing Part 2

In [12]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[12]:
```
Series([], dtype: float64)
```

In [13]:
```
df.shape
```

Out[13]:
```
(4746, 9)
```

Label Encoding for each Object datatype

In [14]:
```
# 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()}")
```

```
Area Type: ['Super Area' 'Carpet Area' 'Built Area']
City: ['Kolkata' 'Mumbai' 'Bangalore' 'Delhi' 'Chennai' 'Hyderabad']
Furnishing Status: ['Unfurnished' 'Semi-Furnished' 'Furnished']
Tenant Preferred: ['Bachelors/Family' 'Bachelors' 'Family']
Point of Contact: ['Contact Owner' 'Contact Agent' 'Contact Builder']
```

In [15]:
```
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()}")
```

```
Area Type: [2 1 0]
City: [4 5 0 2 1 3]
Furnishing Status: [2 1 0]
Tenant Preferred: [1 0 2]
Point of Contact: [2 0 1]
```

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

Out[16]: <AxesSubplot:>

Train Test Split

In [17]:
```
from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Rent', axis=1)
y = df['Rent']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
```

Remove Outlier from train data using Z-Score

In [18]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['Size', 'BHK', 'Bathroom']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree Regressor

In [19]:
```
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'],
    'random_state': [0, 42]
}

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

# 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': 'log2', 'min_samples_leaf': 4, 'min_samples_
split': 2, 'random_state': 0}
```

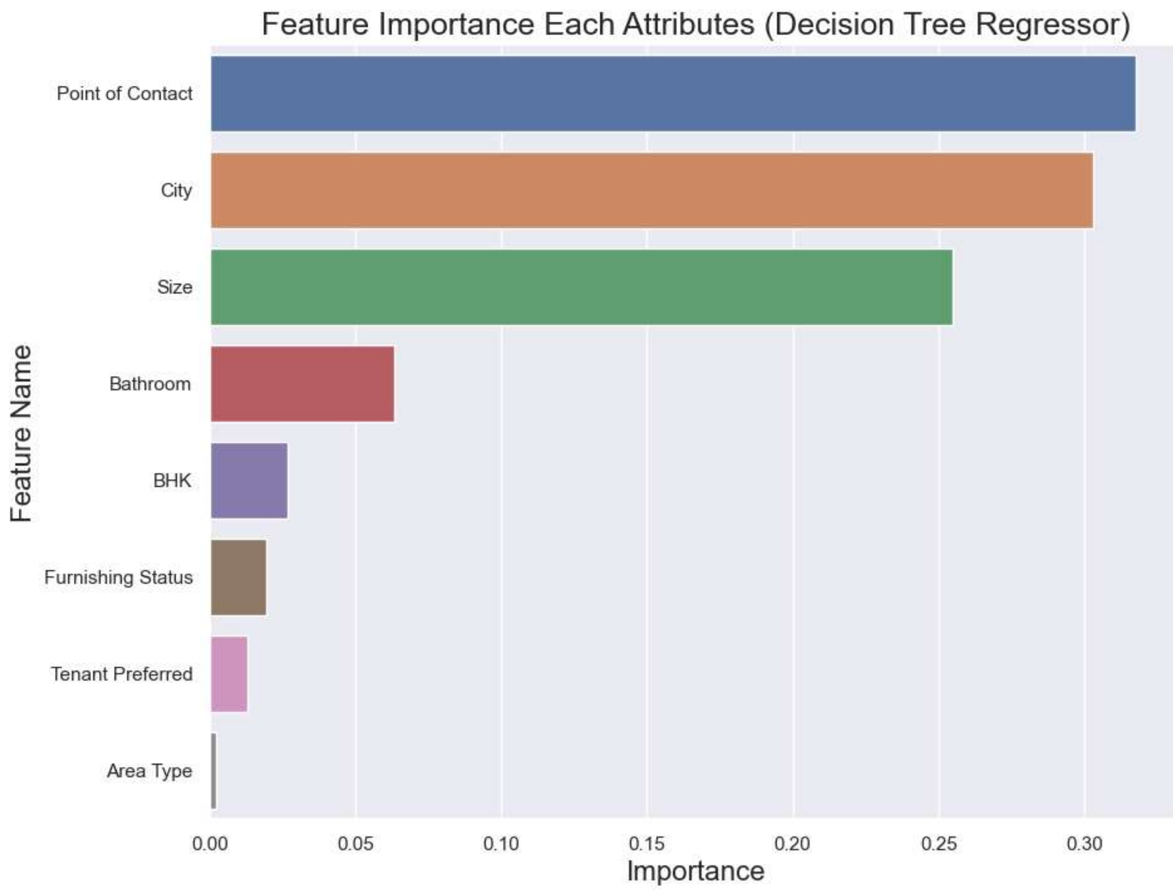
In [20]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=8, max_features='log2'
dtree.fit(X_train, y_train)
```

Out[20]:
```
DecisionTreeRegressor(max_depth=8, max_features='log2', min_samples_leaf=4,
                      random_state=0)
```

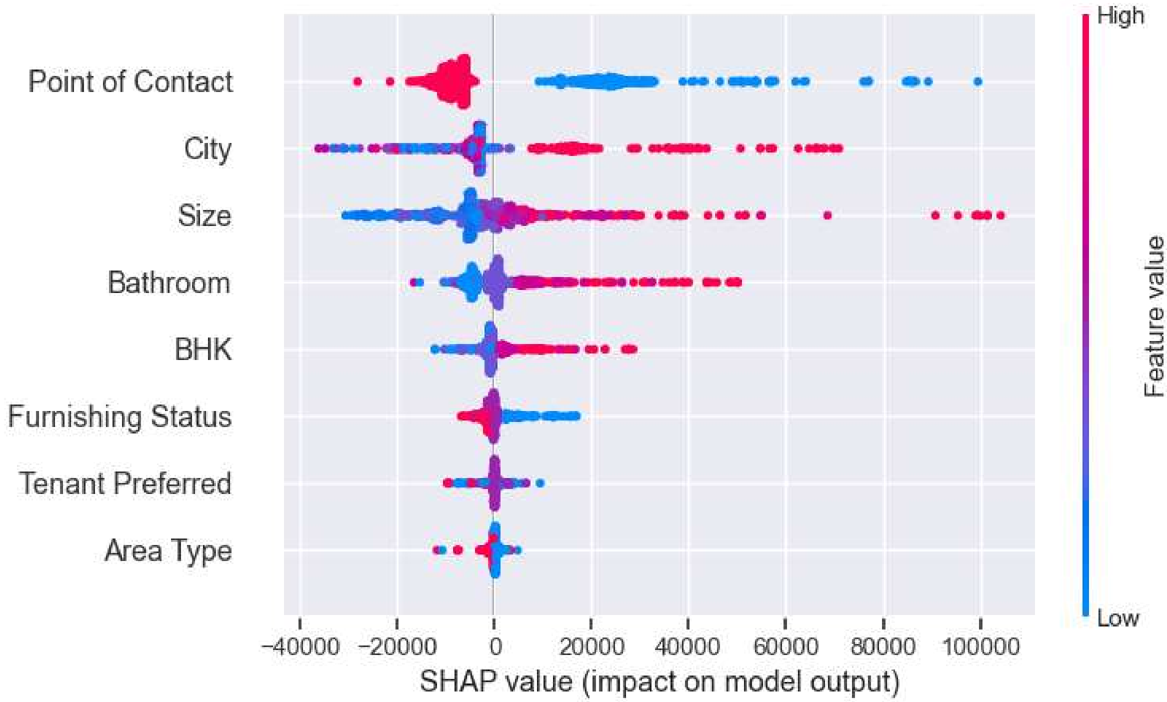
In [21]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = 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 17826.812556914843
MAPE is 0.40664096147987616
MSE is 14292801658.954437
R2 score is 0.20556418893346373
RMSE score is 119552.50586647875
```

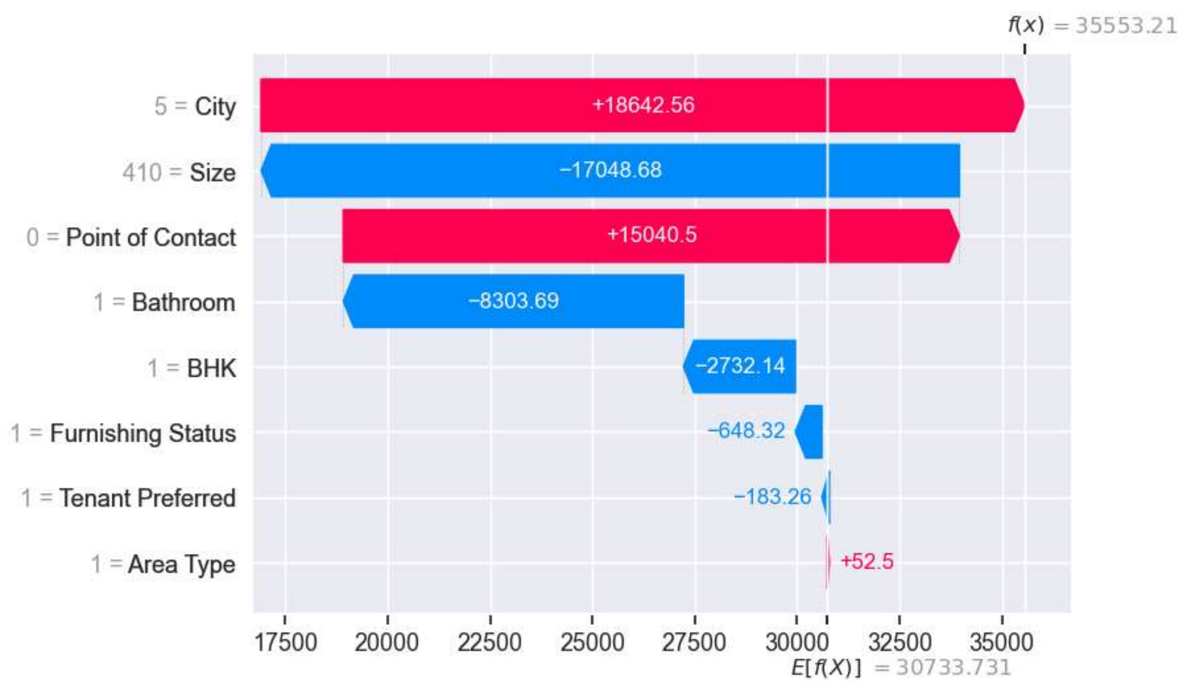
In [22]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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)', font
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [23]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [24]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



Random Forest Regressor

In [25]:
```
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'],
    'random_state': [0, 42]
}

# 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': 'sqrt', 'min_samples_
leaf': 2, 'min_samples_split': 5, 'random_state': 42}
```

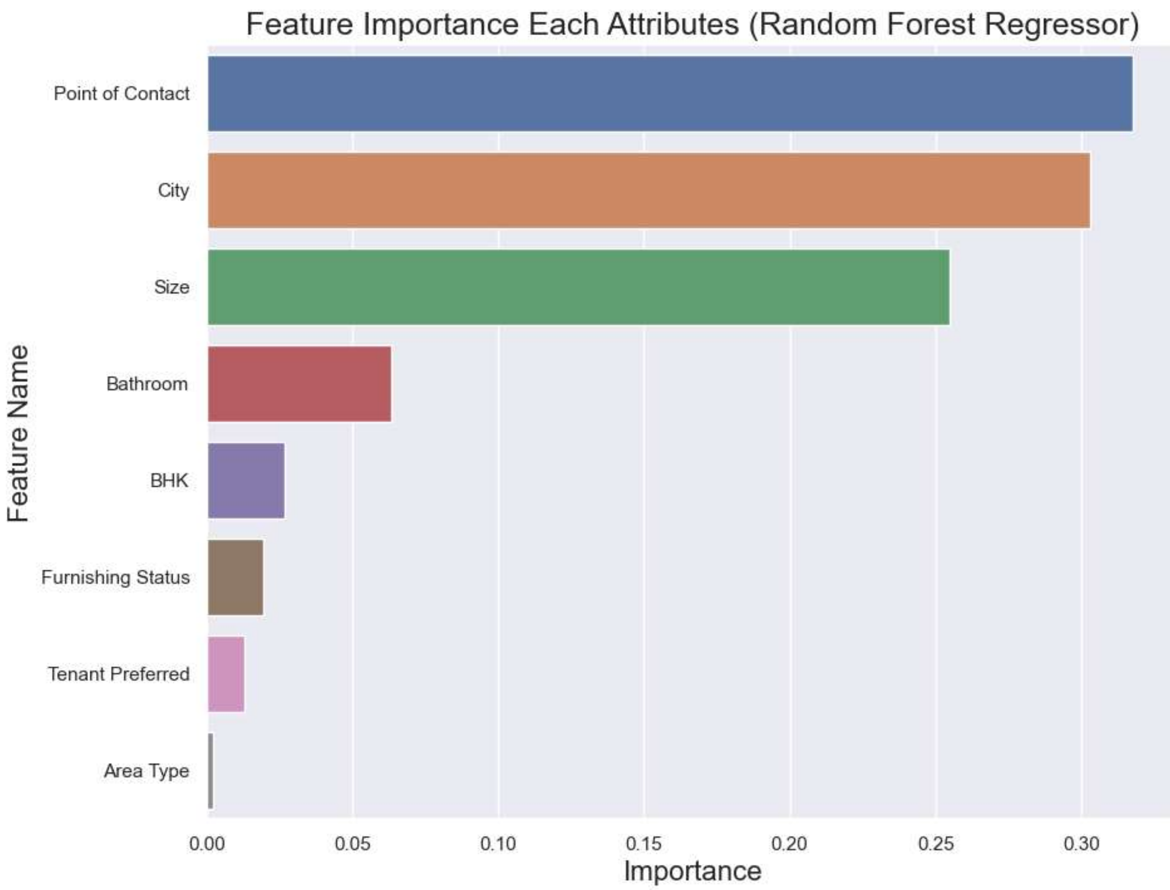
In [26]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42, max_depth=9, min_samples_split=5, 
                           max_features='sqrt')
rf.fit(X_train, y_train)
```

Out[26]:
```
RandomForestRegressor(max_depth=9, max_features='sqrt', min_samples_leaf=2,
                      min_samples_split=5, random_state=42)
```

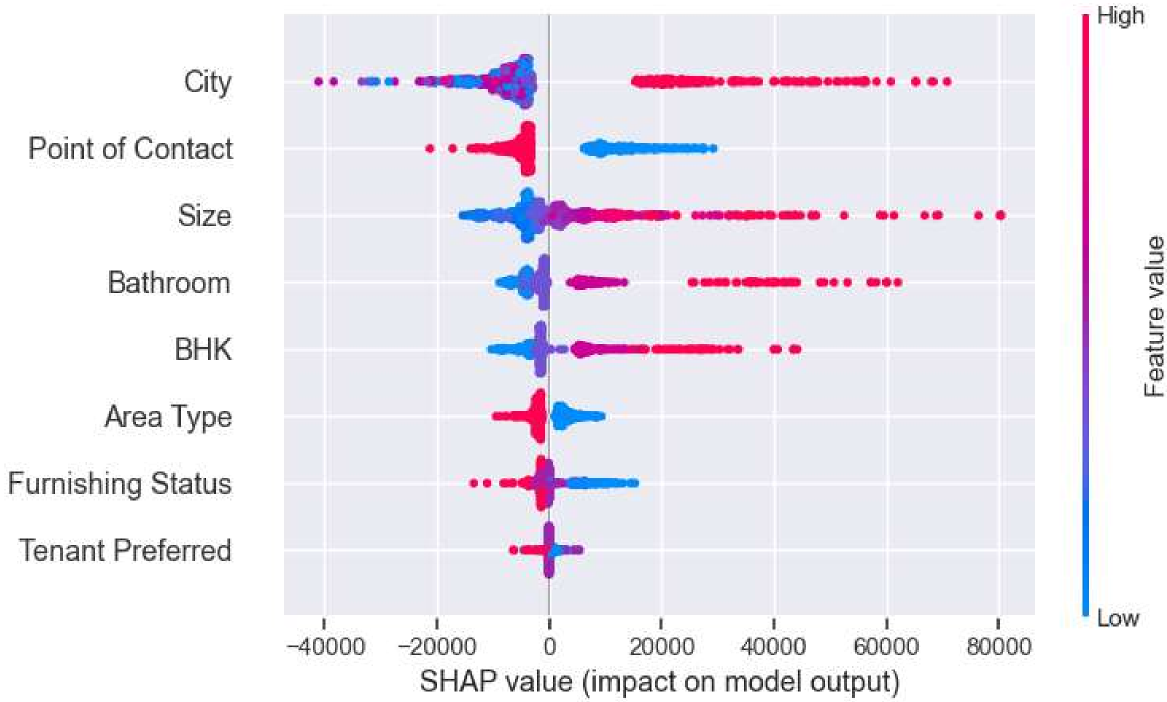
In [27]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, 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 16546.27167682938
MAPE is 0.3655592372864544
MSE is 14083410671.10677
R2 score is 0.21720275380199439
RMSE score is 118673.5466357468
```

In [28]:
```
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)', font
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [29]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [30]:
```
explainer = shap.Explainer(rf, X_test, check_additivity=False)
shap_values = explainer(X_test, check_additivity=False)
shap.plots.waterfall(shap_values[0])
```