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

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

Out[2]:

	Rating	Company Name	Job Title	Salary	Salaries Reported	Location	Employment Status	Job Roles
0	3.8	Sasken	Android Developer	400000	3	Bangalore	Full Time	Android
1	4.5	Advanced Millennium Technologies	Android Developer	400000	3	Bangalore	Full Time	Android
2	4.0	Unacademy	Android Developer	1000000	3	Bangalore	Full Time	Android
3	3.8	SnapBizz Cloudtech	Android Developer	300000	3	Bangalore	Full Time	Android
4	4.4	Appoids Tech Solutions	Android Developer	600000	3	Bangalore	Full Time	Android

Data Preprocessing Part 1

In [3]:
```
df.dtypes
```

Out[3]:
```
Rating                 float64
Company Name            object
Job Title               object
Salary                   int64
Salaries Reported        int64
Location                object
Employment Status       object
Job Roles               object
dtype: object
```

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

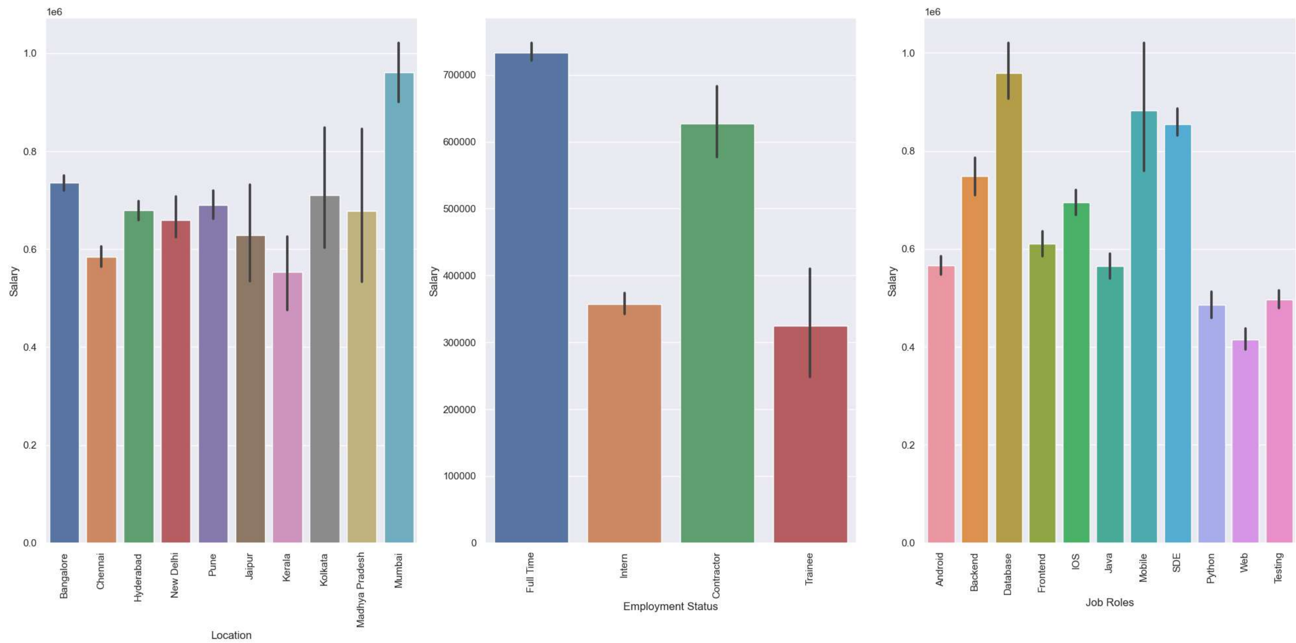
Out[4]:
```
Company Name         11261
Job Title             1080
Location                10
Employment Status        4
Job Roles               11
dtype: int64
```

In [5]:
```
df['Job Roles'].unique()
```

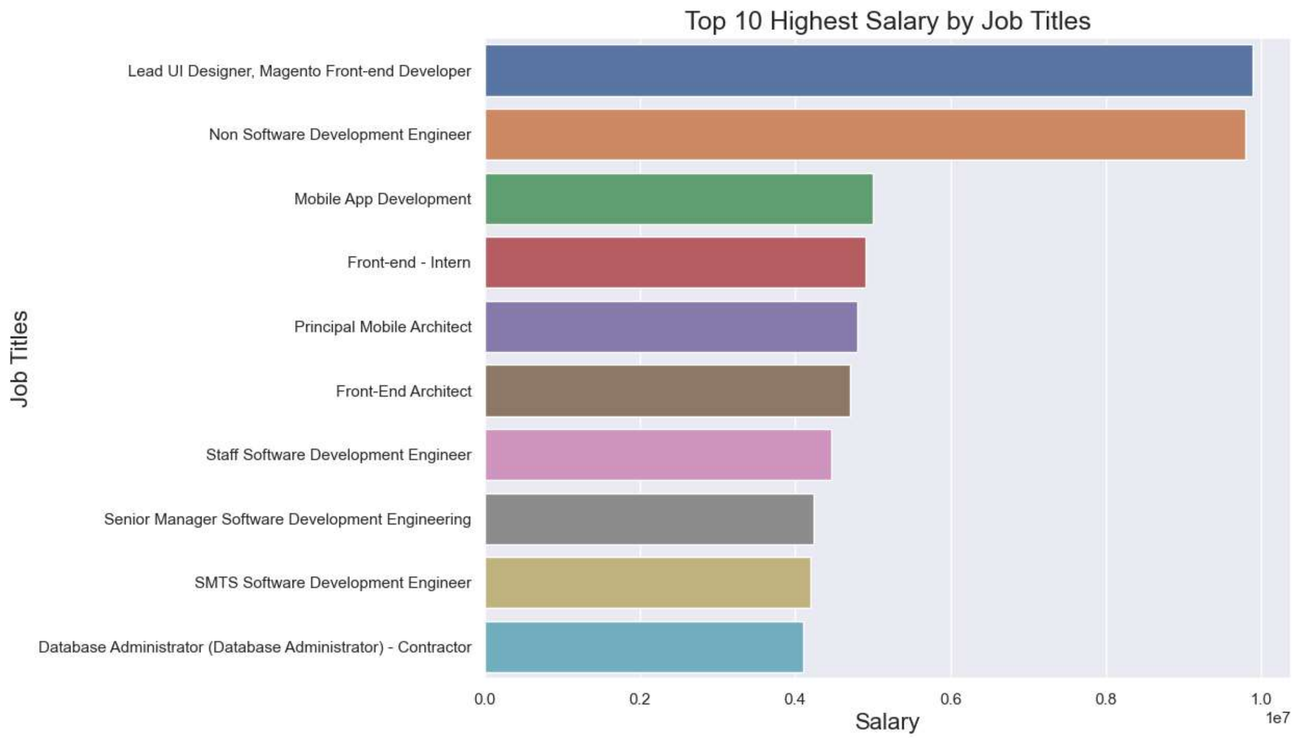
Out[5]:
```
array(['Android', 'Backend', 'Database', 'Frontend', 'IOS', 'Java',
       'Mobile', 'SDE', 'Python', 'Web', 'Testing'], dtype=object)
```

Exploratory Data Analysis

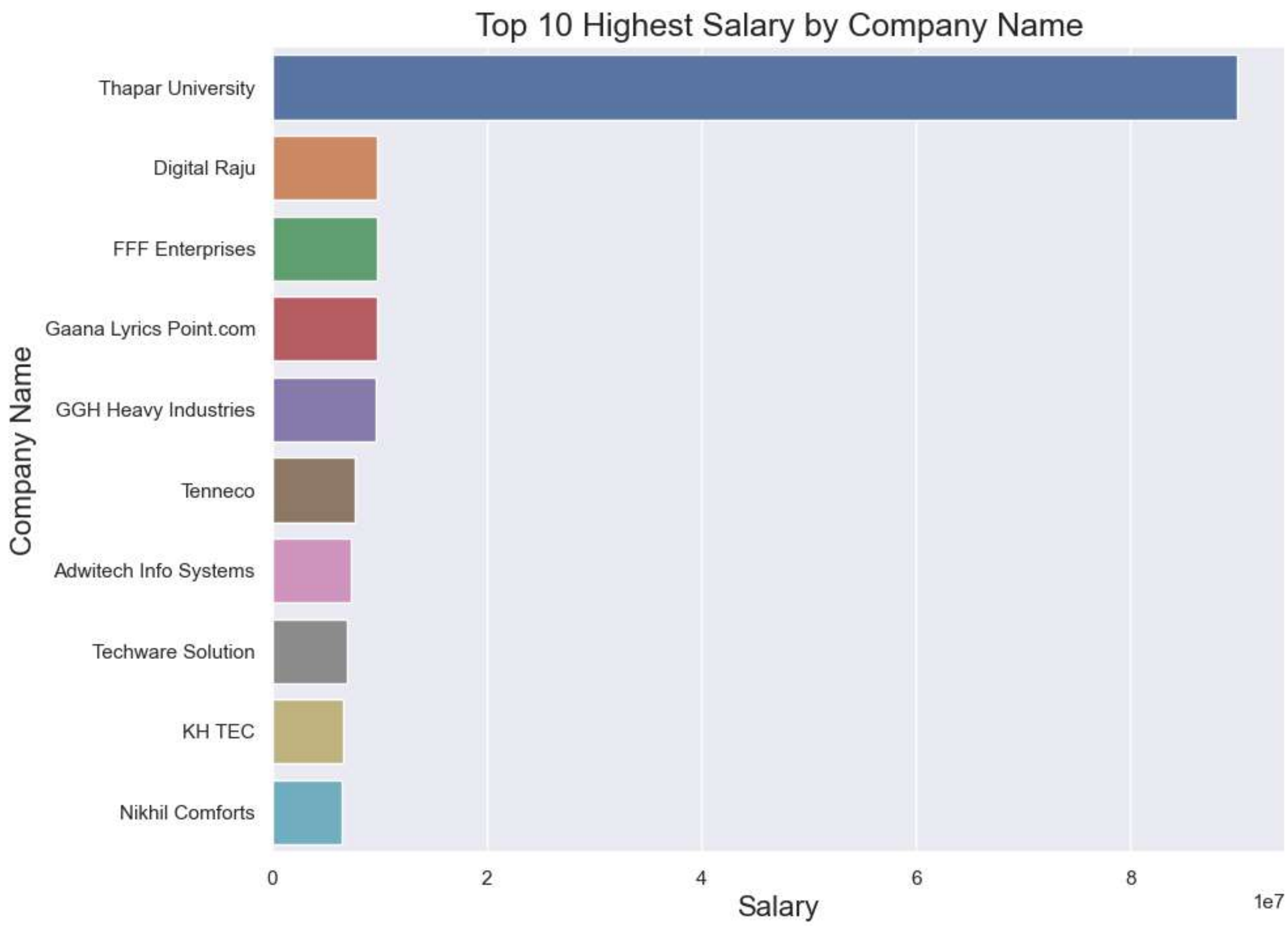
In [6]:
```
# list of categorical variables to plot
cat_vars = ['Location', 'Employment Status', 'Job Roles']

# create figure with subplots
fig, axs = plt.subplots(nrows=1, 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='Salary', 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()

# show plot
plt.show()
```



In [7]:
```
dfjob = df.groupby('Job Title', as_index=False)['Salary'].mean()
dfjob2 = dfjob.nlargest(10,'Salary')

#Show Bar Chart
plt.figure(figsize=(10,8))
sns.barplot(data=dfjob2, x='Salary', y='Job Title')
plt.title('Top 10 Highest Salary by Job Titles', fontsize=18)
plt.xlabel ('Salary', fontsize=16)
plt.ylabel ('Job Titles', fontsize=16)
plt.show()
```

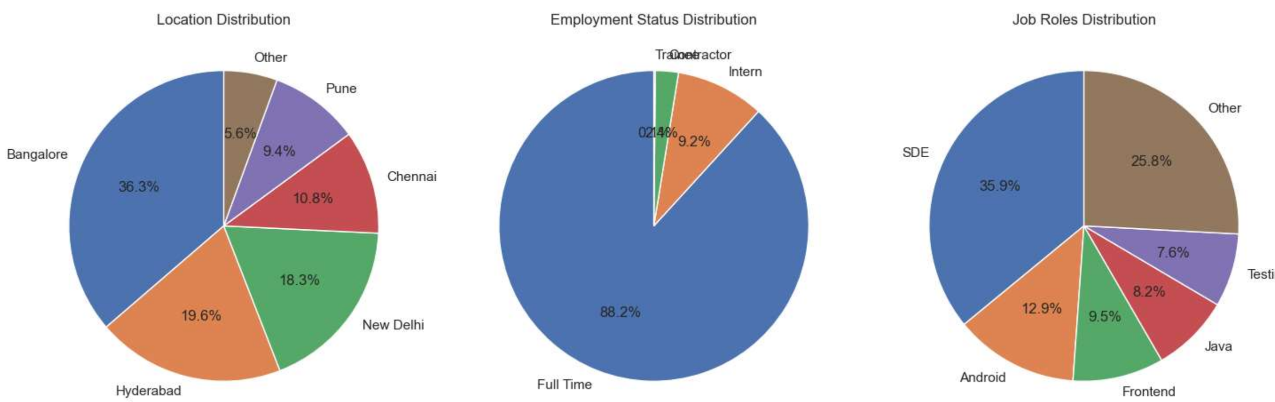


In [8]:
```
# Group by company_location and calculate the mean salary
dfcom = df.groupby('Company Name')['Salary'].mean().reset_index()

# Get the top 10 highest salaries by company_location
dfcom2 = dfcom.nlargest(10, 'Salary')

# Show Bar Chart
plt.figure(figsize=(10, 8))
sns.barplot(data=dfcom2, x='Salary', y='Company Name')
plt.title('Top 10 Highest Salary by Company Name', fontsize=18)
plt.xlabel('Salary', fontsize=16)
plt.ylabel('Company Name', fontsize=16)
plt.show()
```

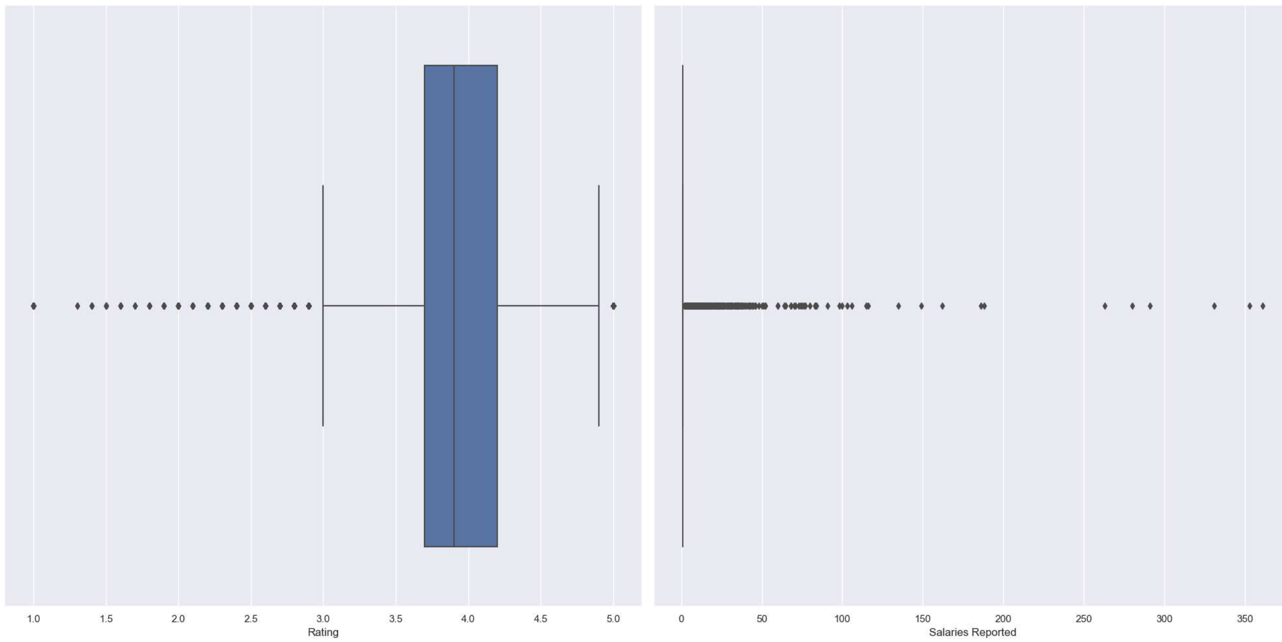


In [9]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

cat_vars = ['Location', 'Employment Status', 'Job Roles']

# Create a figure and axes
fig, axs = plt.subplots(nrows=1, 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(), index=['Other'])
            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%%', startangle=90)

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

# Adjust spacing between subplots
fig.tight_layout()

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

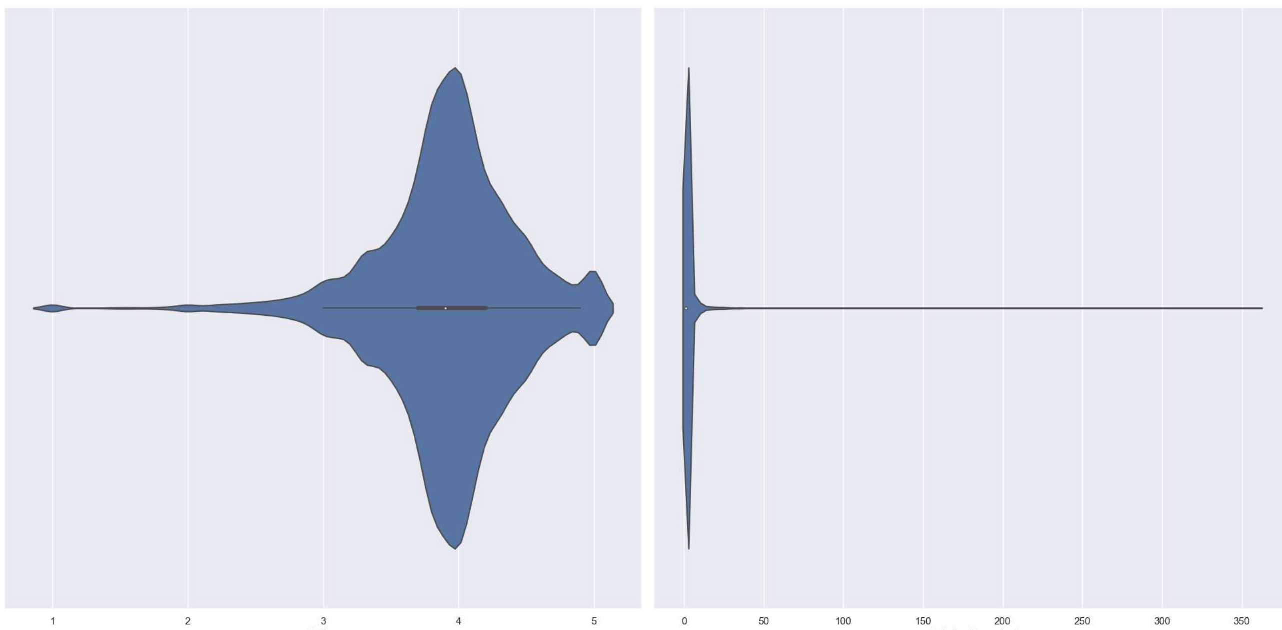
```
C:\Users\Michael\AppData\Local\Temp\ipykernel_2804\3255518657.py:19: FutureWarning: The series.append method is depre
cated and will be removed from pandas in a future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_2804\3255518657.py:19: FutureWarning: The series.append method is depre
cated 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]:
```
num_vars = ['Rating', 'Salaries Reported']

fig, axs = plt.subplots(nrows=1, ncols=2, 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 = ['Rating', 'Salaries Reported']

fig, axs = plt.subplots(nrows=1, ncols=2, 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]:
```
(22770, 8)
```

In [14]:
```
df.head()
```

Out[14]:

	Rating	Company Name	Job Title	Salary	Salaries Reported	Location	Employment Status	Job Roles
0	3.8	Sasken	Android Developer	400000	3	Bangalore	Full Time	Android
1	4.5	Advanced Millennium Technologies	Android Developer	400000	3	Bangalore	Full Time	Android
2	4.0	Unacademy	Android Developer	1000000	3	Bangalore	Full Time	Android
3	3.8	SnapBizz Cloudtech	Android Developer	300000	3	Bangalore	Full Time	Android
4	4.4	Appoids Tech Solutions	Android Developer	600000	3	Bangalore	Full Time	Android

In [15]:
```
# Drop columns that have alot of unique value like Company Name and Job Title because its irrelevant
df.drop(columns=['Company Name', 'Job Title'], inplace=True)
df.head()
```

Out[15]:

	Rating	Salary	Salaries Reported	Location	Employment Status	Job Roles
0	3.8	400000	3	Bangalore	Full Time	Android
1	4.5	400000	3	Bangalore	Full Time	Android
2	4.0	1000000	3	Bangalore	Full Time	Android
3	3.8	300000	3	Bangalore	Full Time	Android
4	4.4	600000	3	Bangalore	Full Time	Android

Label Encoding for each Object datatype

In [16]:
```
# 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()}")
```

```
Location: ['Bangalore' 'Chennai' 'Hyderabad' 'New Delhi' 'Pune' 'Jaipur' 'Kerala'
 'Kolkata' 'Madhya Pradesh' 'Mumbai']
Employment Status: ['Full Time' 'Intern' 'Contractor' 'Trainee']
Job Roles: ['Android' 'Backend' 'Database' 'Frontend' 'IOS' 'Java' 'Mobile' 'SDE'
 'Python' 'Web' 'Testing']
```

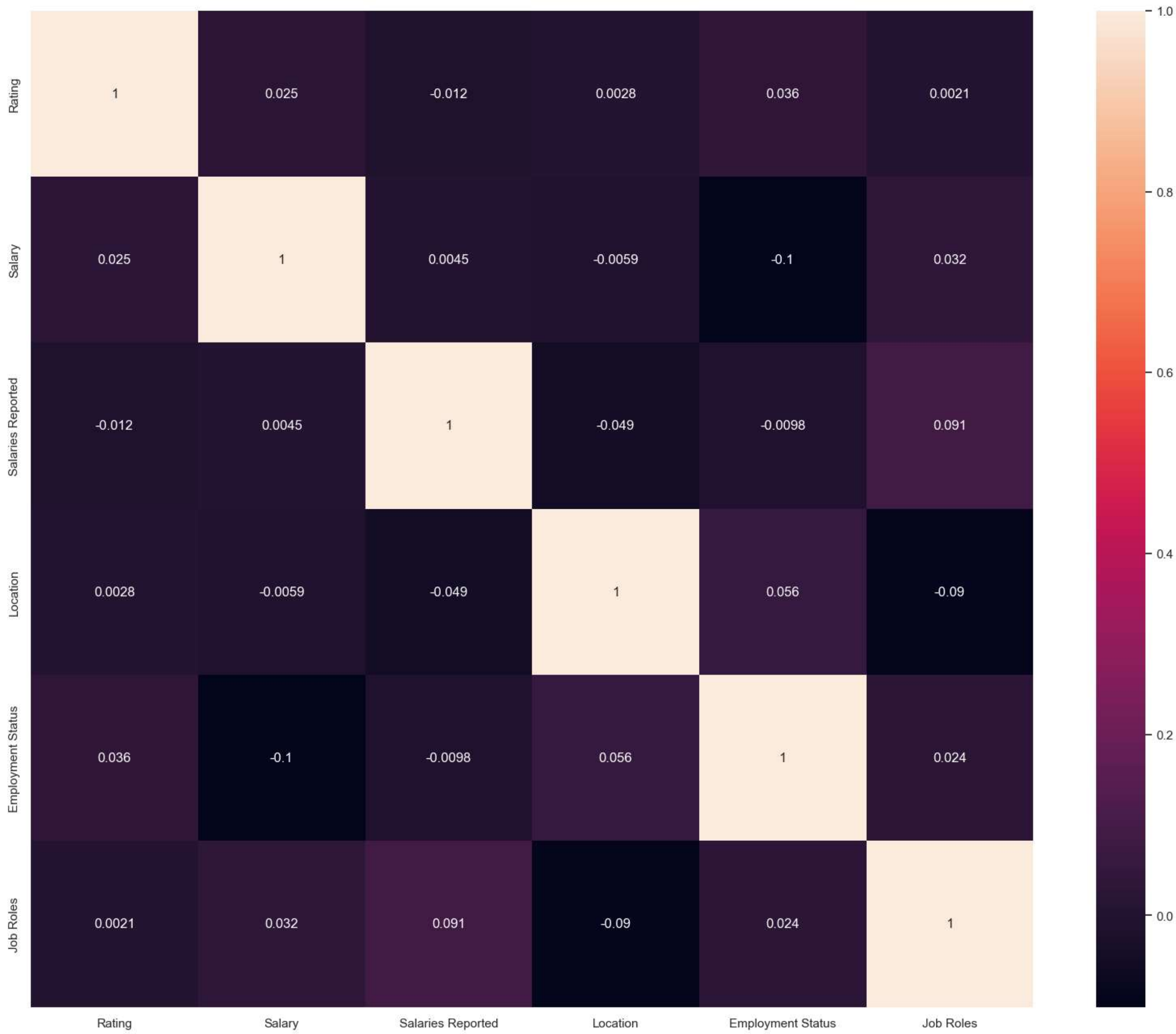
In [17]:
```
from sklearn import preprocessing

# 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()}")
```

```
Location: [0 1 2 8 9 3 4 5 6 7]
Employment Status: [1 2 0 3]
Job Roles: [ 0  1  2  3  4  5  6  8  7 10  9]
```

Correlation Heatmap

In [18]:
```
#Correlation Heatmap (print the correlation score each variables)
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[18]: <AxesSubplot:>



Train Test Split

In [19]:
```
from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Salary', axis=1)
y = df['Salary']

# 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, random_state=0)
```

Remove Outlier from Train data using IQR

In [20]:
```
# 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('Salary', axis=1)
y_train = train_df['Salary']
```

Decision Tree Regressor

In [21]:
```
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_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': 3, 'min_samples_split': 2, 'random_state': 0}
```

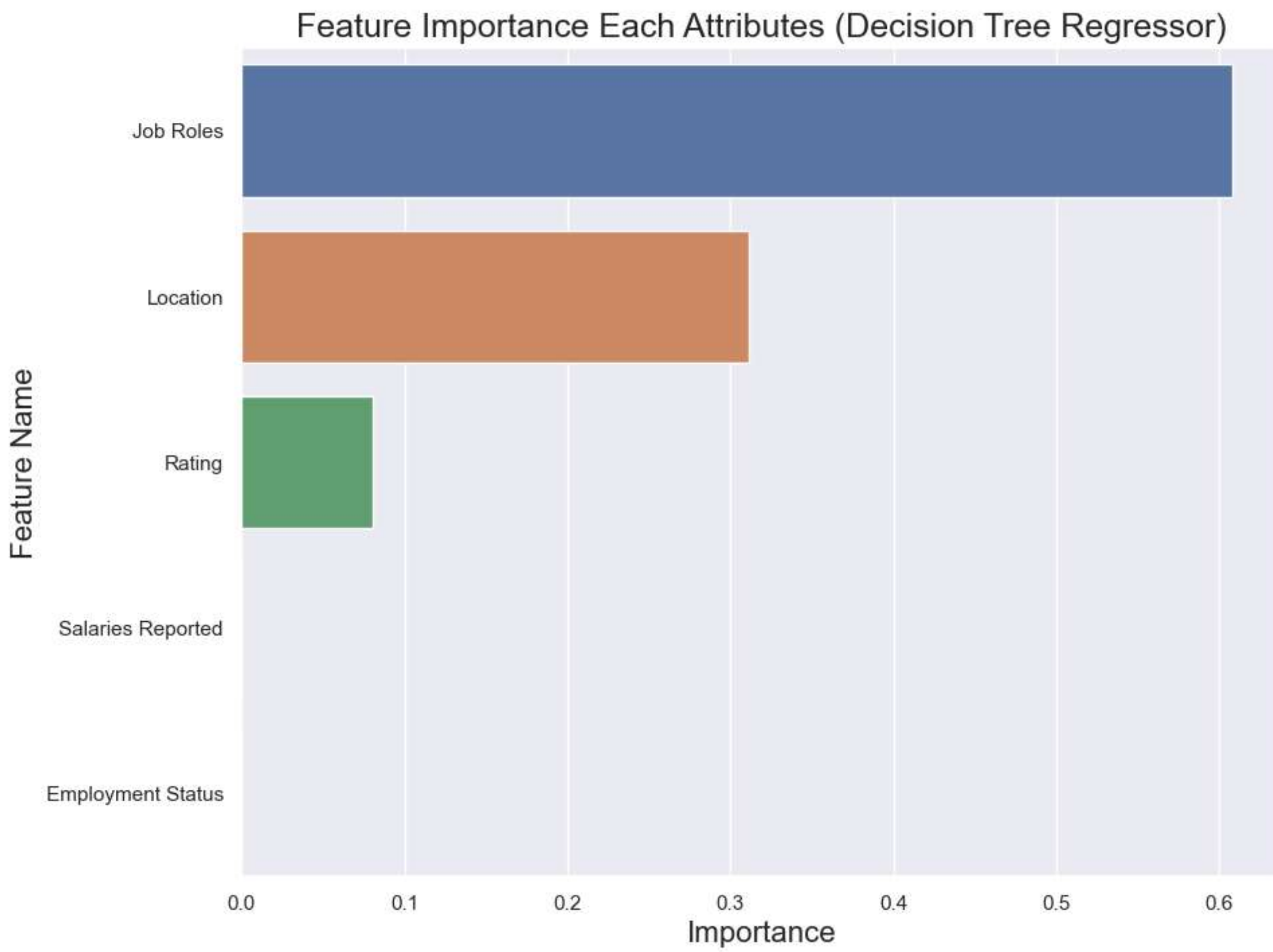
In [22]:
```
from sklearn.tree import DecisionTreeRegressor
dtree = DecisionTreeRegressor(random_state=0, max_depth=6, max_features='auto', min_samples_leaf=3, min_samples_split=
dtree.fit(X_train, y_train)
```

Out[22]:
```
DecisionTreeRegressor(max_depth=6, max_features='auto', min_samples_leaf=3,
                      random_state=0)
```

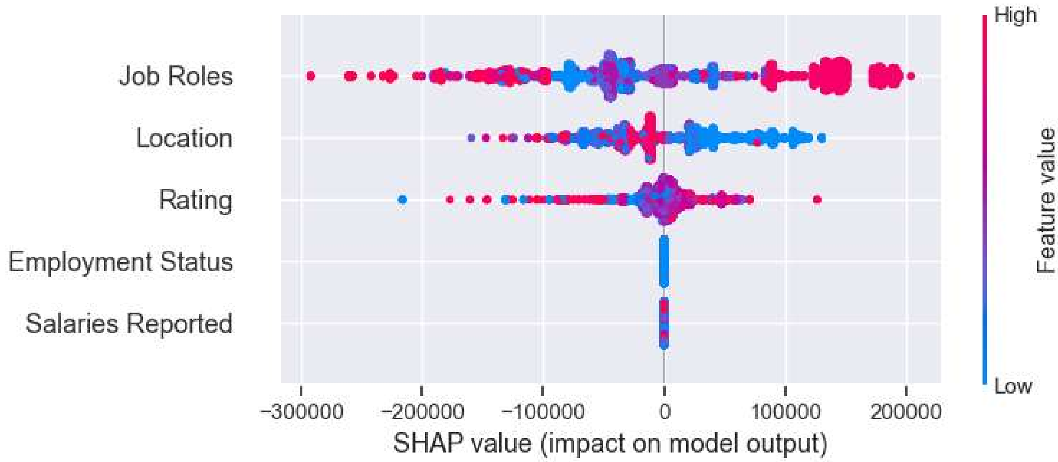
In [23]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = dtree.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = 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 406690.85774989496
MAPE is 1.1597214620682048
MSE is 453645939061.63
R2 score is 0.05865331750696001
RMSE score is 673532.4335632472
```

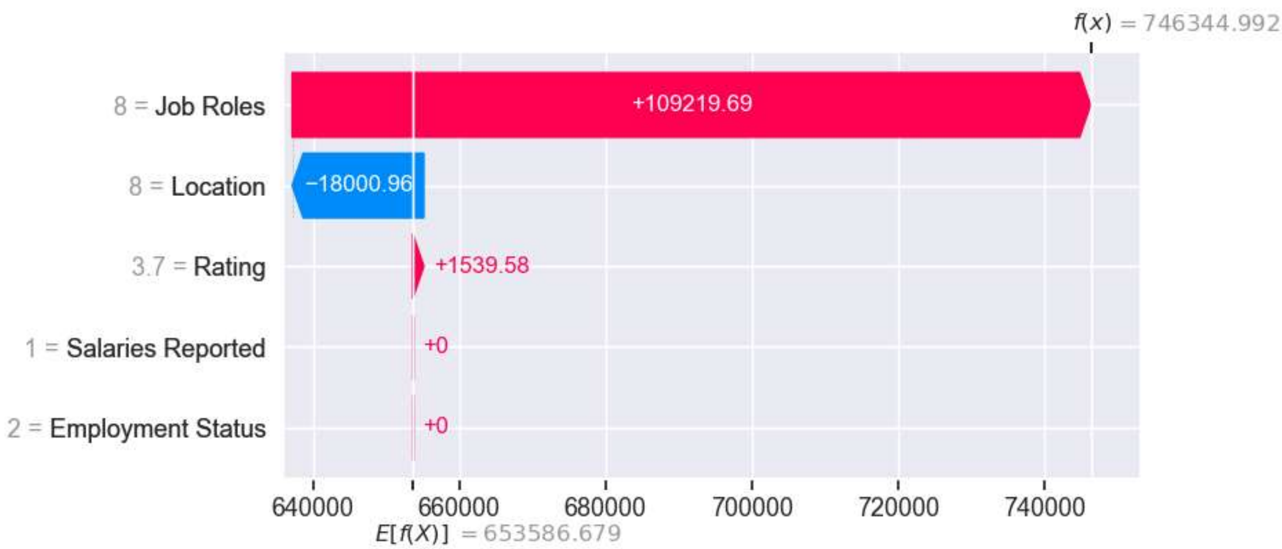
In [24]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 [25]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [26]:
```
explainer = shap.Explainer(dtree, X_test)
shap_values = explainer(X_test)
shap.plots.waterfall(shap_values[0])
```



Random Forest Regressor

In [27]:
```
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': 7, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 2, 'rando
m_state': 0}
```

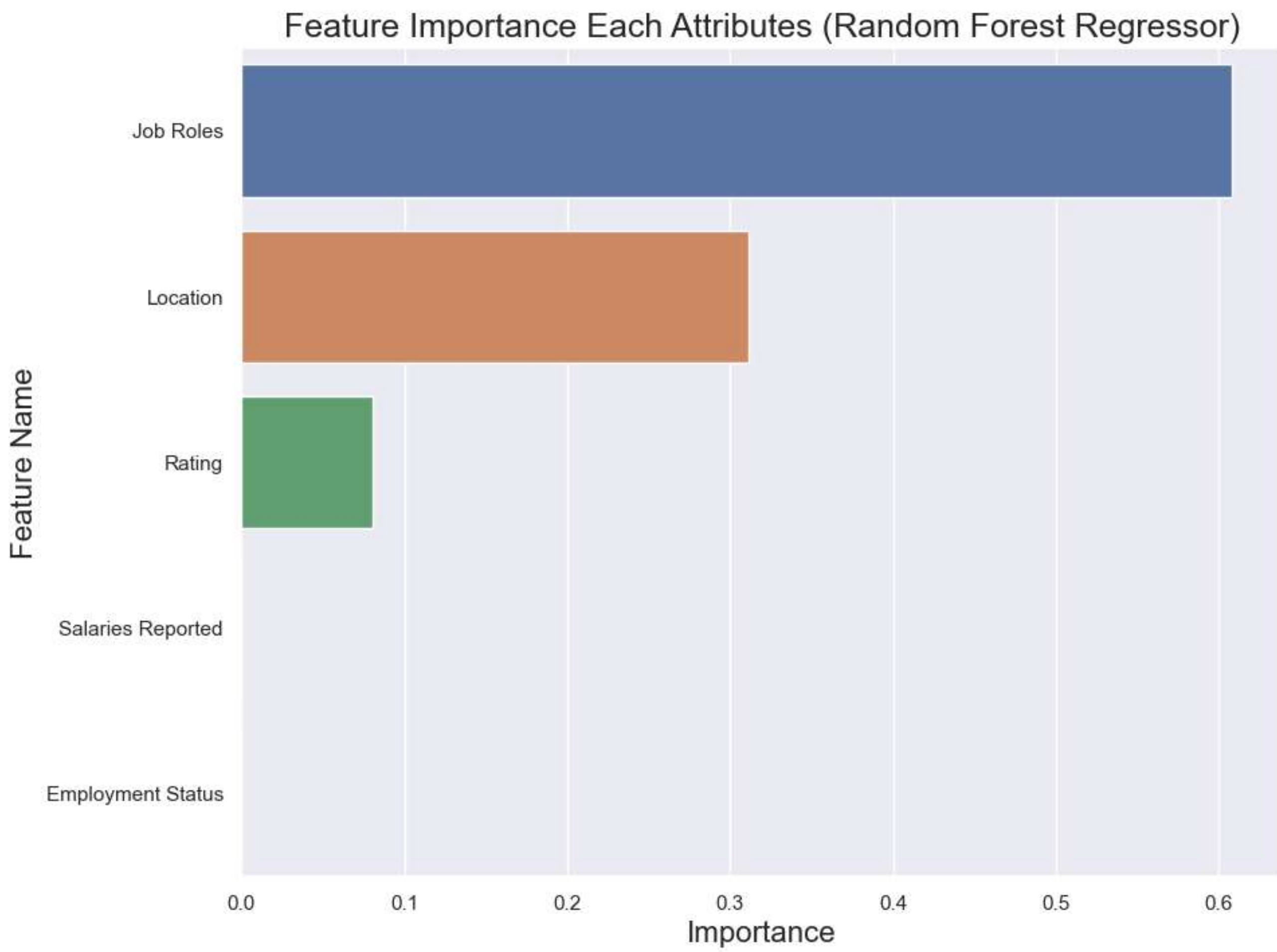
In [28]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=0, max_depth=7, min_samples_split=2, min_samples_leaf=4,
                           max_features='sqrt')
rf.fit(X_train, y_train)
```

Out[28]:
```
RandomForestRegressor(max_depth=7, max_features='sqrt', min_samples_leaf=4,
                      random_state=0)
```

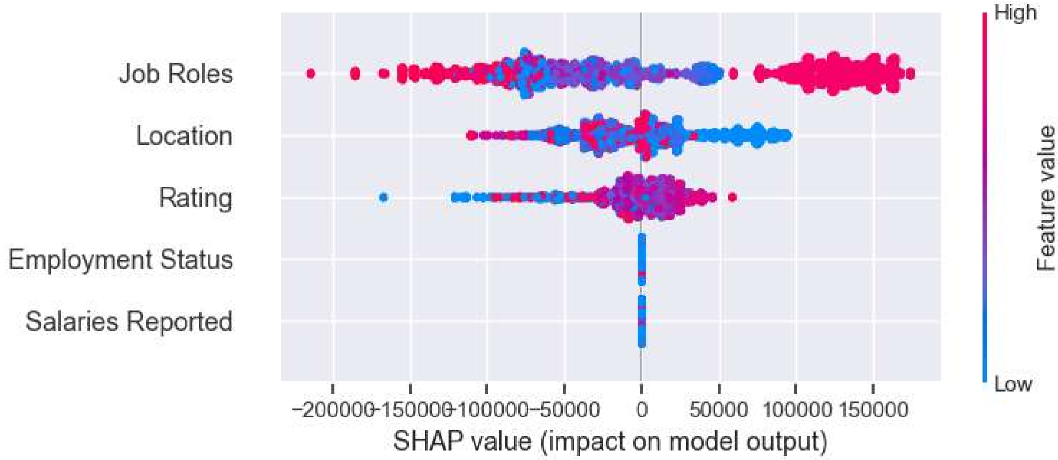
In [29]:
```
from sklearn import metrics
from sklearn.metrics import mean_absolute_percentage_error
import math
y_pred = rf.predict(X_test)
mae = metrics.mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
mse = metrics.mean_squared_error(y_test, y_pred)
r2 = metrics.r2_score(y_test, 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 406528.3702832092
MAPE is 1.1496807707475947
MSE is 455093486741.35516
R2 score is 0.05564955600767607
RMSE score is 674606.1715855816
```

In [30]:
```
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 [31]:
```
import shap
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



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

```
98%|==================| 4472/4554 [00:24<00:00]
```