Name - Abhishek Jani
Roll No - 16

DSPL PROJECT REPORT

Topic - House Rent Prediction System

Problem Statement -

The House Rent price Prediction System strives to locate the best properties in your neighborhood with the most affordable rental rates.

The House Rent Prediction System aims to address these issues by providing accurate and transparent rent predictions based on relevant factors. This will help landlords set rents that are in line with market rates, and tenants make more informed decisions about their housing options.

Data Preprocessing-

1. Cleaning:

In [1]:
```
#importing the pandas and numpy library
import pandas as pd
import numpy as np
```

In [12]:
```
#reading the csv file
dataframe = pd.read_csv("House_Rent_main1.csv")
```

In [13]:
```
#printing the number of samples and attributes of dataset
print(dataframe.shape)
```

```
(4746, 12)
```

In [22]:
```
dataframe.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4746 entries, 0 to 4745
Data columns (total 12 columns):
 #   Column             Non-Null Count  Dtype 
---  ------             --------------  ----- 
 0   Posted On          4746 non-null   object
 1   BHK                4746 non-null   int64 
 2   Rent               4746 non-null   int64 
 3   Size               4746 non-null   int64 
 4   Floor              4746 non-null   object
 5   Area Type          4746 non-null   object
 6   Area Locality      4746 non-null   object
 7   City               4746 non-null   object
 8   Furnishing Status  4746 non-null   int64 
 9   Tenant Preferred   4746 non-null   object
 10  Bathroom           4746 non-null   int64 
 11  Point of Contact   4746 non-null   object
dtypes: int64(5), object(7)
memory usage: 445.1+ KB
```

1.a) Head of the dataset:

In [37]:
```
#print the first 5 samples
print(dataframe.head())
```

```
   BHK   Rent  Size            Floor  Area Type              Area Locality  \
0    2  10000  1100  Ground out of 2          1                     Bandel
1    2  20000   800       1 out of 3          1  Phool Bagan, Kankurgachi
2    2  17000  1000       1 out of 3          1   Salt Lake City Sector 2
3    2  10000   800       1 out of 2          1                Dumdum Park
4    2   7500   850       1 out of 2          2             South Dum Dum

      City  Furnishing Status  Tenant Preferred  Bathroom
0  Kolkata                  1  Bachelors/Family         2
1  Kolkata                  2  Bachelors/Family         1
2  Kolkata                  2  Bachelors/Family         1
3  Kolkata                  1  Bachelors/Family         1
4  Kolkata                  1         Bachelors         1
```

1.b) Missing Values:

In [23]:
```
#New dataframe
new_df = dataframe
#Checking for null values
print(new_df.isnull().sum())
print("Missing values distribution: ")
print(new_df.isnull().mean())
```

```
Posted On            0
BHK                  0
Rent                 0
Size                 0
Floor                0
Area Type            0
Area Locality        0
City                 0
Furnishing Status    0
Tenant Preferred     0
Bathroom             0
Point of Contact     0
dtype: int64
Missing values distribution:
Posted On            0.0
BHK                  0.0
Rent                 0.0
Size                 0.0
Floor                0.0
Area Type            0.0
Area Locality        0.0
City                 0.0
Furnishing Status    0.0
Tenant Preferred     0.0
Bathroom             0.0
```

1.c) Changing the string values to integers:

In [38]:
```
# changing the attributes of dataset for training prupose
new_dataframe.replace({"Super Area" : "1" , "Carpet Area": "2"},inplace = True)
print(new_dataframe.shape)
new_dataframe
```

```
(4746, 12)
```

Out[38]:

	Posted On	BHK	Rent	Size	Floor	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred	Bathroom
0	2022-05-18	2	10000	1100	Ground out of 2	1	Bandel	Kolkata	1	Bachelors/Family	2
1	2022-05-13	2	20000	800	1 out of 3	1	Phool Bagan, Kankurgachi	Kolkata	2	Bachelors/Family	1

```
In [39]: # replacing the Furninshing status
new_dataframe.replace({"Unfurnished" : "1" , "Furnished": "2" , "Semi-Furnished" : "2"},inplace = True)
print(new_dataframe.shape)
new_dataframe
```

(4746, 12)

Out[39]:

	Posted On	BHK	Rent	Size	Floor	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred	Bathroom
0	2022-05-18	2	10000	1100	Ground out of 2	1	Bandel	Kolkata	1	Bachelors/Family	2
1	2022-05-13	2	20000	800	1 out of 3	1	Phool Bagan, Kankurgachi	Kolkata	2	Bachelors/Family	1

1.d) Finding Duplicates:

```
In [18]: #finding duplicate in dataset
duplicate = new_dataframe.duplicated()
# print(duplicate)
# finding dupicate oin particular column
rent = new_dataframe.Rent.duplicated()
print(rent)
print(new_dataframe.Size.duplicated())
# finding any duplicate value present in dataset --> it will return false is there is no duplicate value
print(new_dataframe.duplicated().any())
```

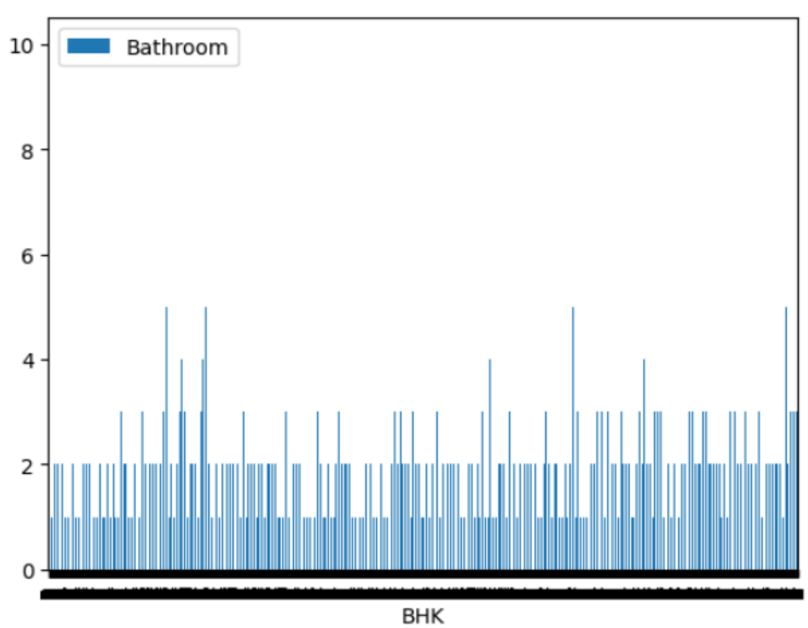
```
0       False
1       False
2       False
3        True
4       False
        ...
4741     True
4742     True
```

2. Data Visualization:

2.a) Bar Graph

```
In [8]: #plotting a bar graph
import matplotlib.pyplot as plt
new_dataframe = dataframe
new_dataframe.plot(x="BHK", y="Bathroom", kind="bar")
```

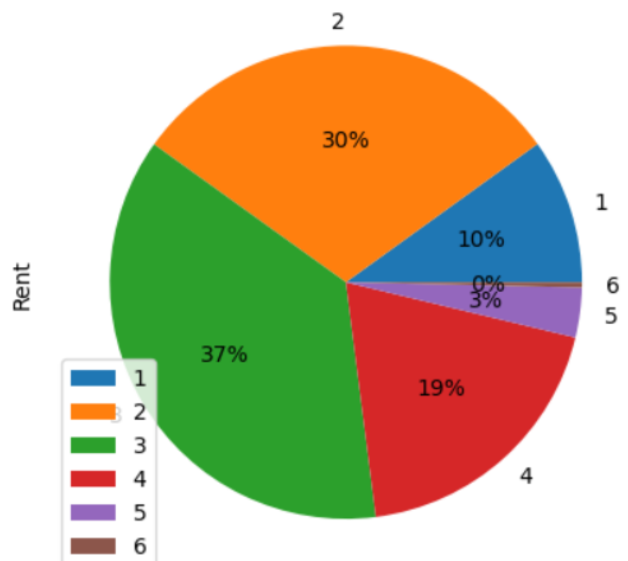
Out[8]: <AxesSubplot:xlabel='BHK'>

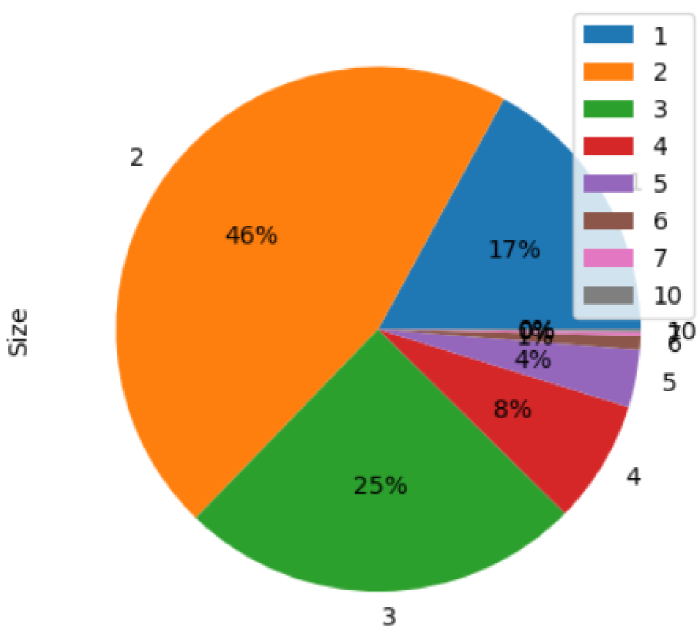


2.b) Pie Chart

```
In [15]: #plotting a pie chart
         import matplotlib.pyplot as plt
         # Plotting the pie chart for above dataframe
         new_dataframe.groupby(['BHK']).sum().plot(kind='pie', y='Rent', autopct='%1.0f%%')
         new_dataframe.groupby(['Bathroom']).sum().plot(kind='pie', y='Size', autopct='%1.0f%%')
```

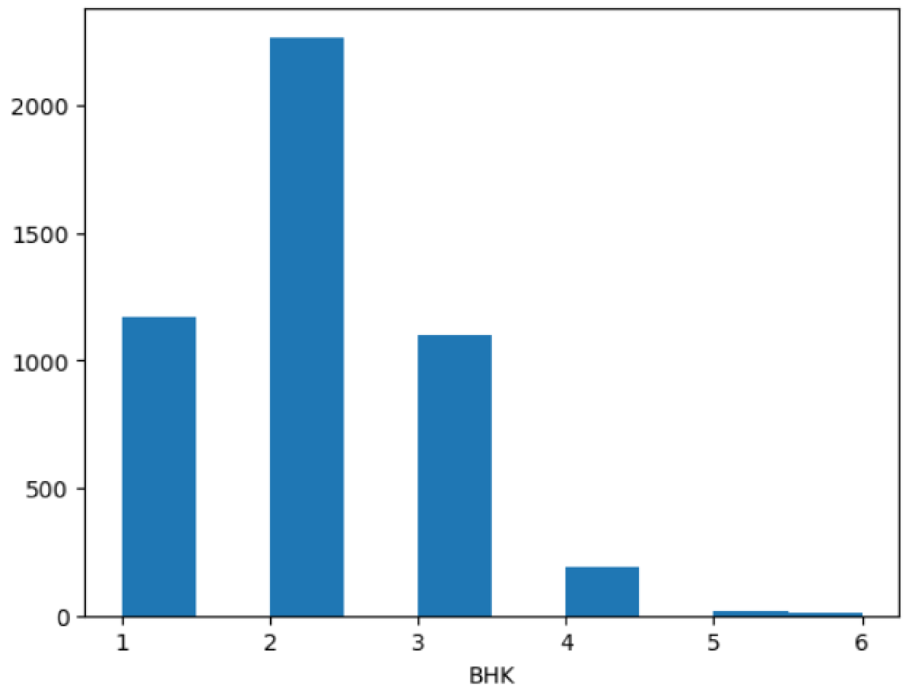
```
Out[15]: <AxesSubplot:ylabel='Size'>
```





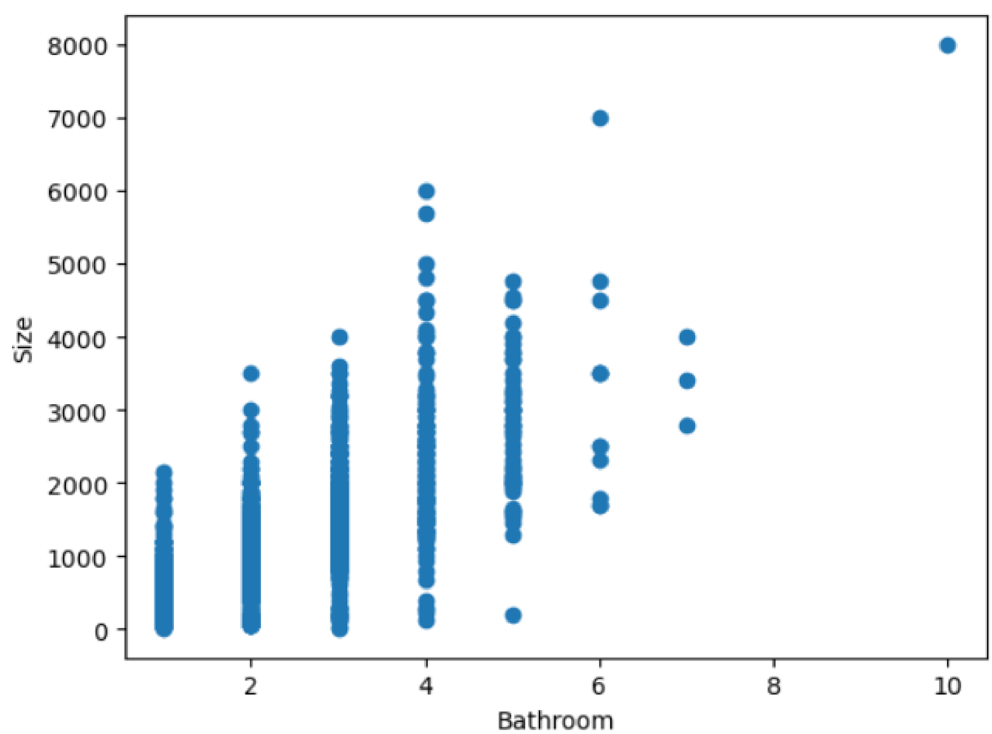
2.c) Univariate Histogram -

```
In [25]: #plotting a histogram for number's of BHK's
         plt.hist(new_dataframe['BHK'])
         plt.xlabel("BHK")

Out[25]: Text(0.5, 0, 'BHK')
```



2.d) Scatter Plot

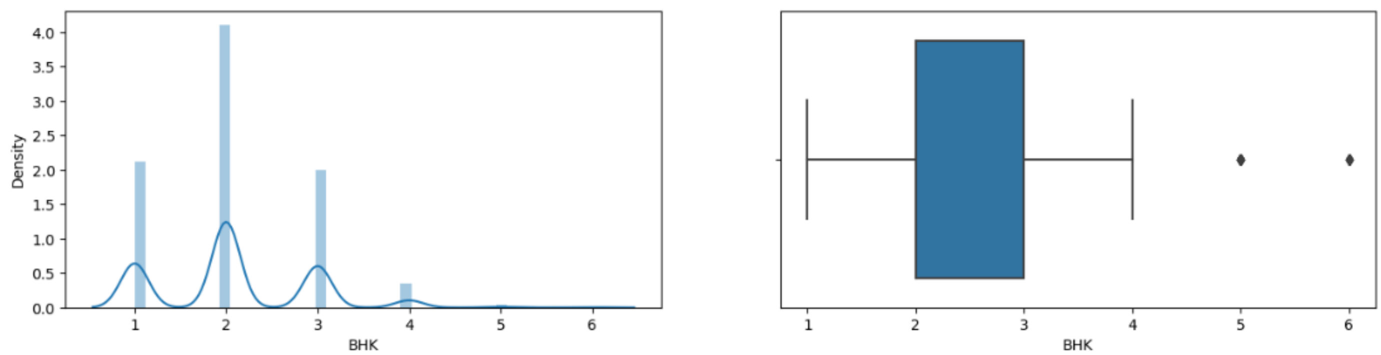
```
In [28]: # Syntax of scatter plot()
         plt.scatter(new_dataframe['Bathroom'], new_dataframe['Size'])
         plt.xlabel("Bathroom")
         plt.ylabel("Size")
         plt.show()
```



3.Removing the outliers

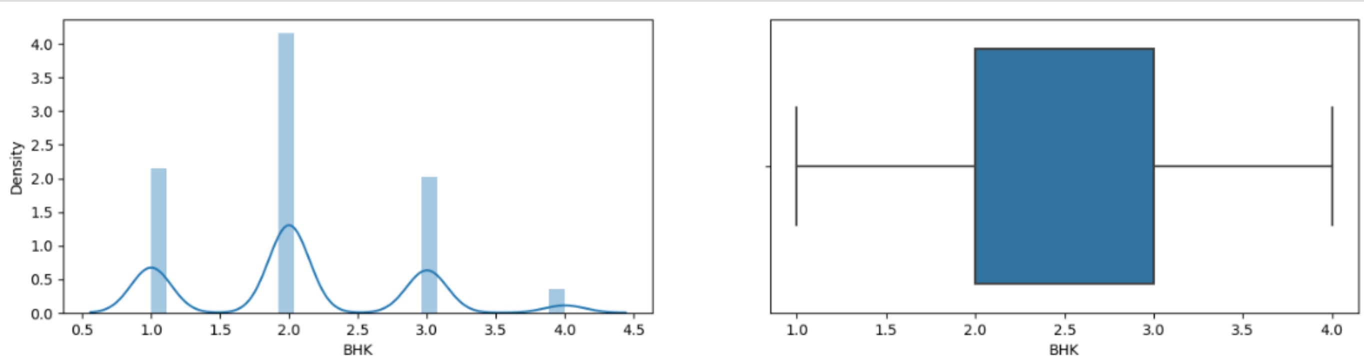
3.a) Before Trimming:

In [68]:
```
# compare plots before trimming
plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df1['BHK'])
plt.subplot(2,2,2)
sns.boxplot(df1['BHK'])
plt.show()
```



3.b) After Trimming:

In [72]:
```
# compare plots after trimming
plt.figure(figsize=(16,8))
plt.subplot(2,2,1)
sns.distplot(df2['BHK'])
plt.subplot(2,2,2)
sns.boxplot(df2['BHK'])
plt.show()
```



4.Handling Class Imbalance :

Handling class imbalance refers to the techniques and strategies used to address the problem of imbalanced class distributions in a dataset, where one class (the minority class) has significantly fewer instances than another class (the majority class).

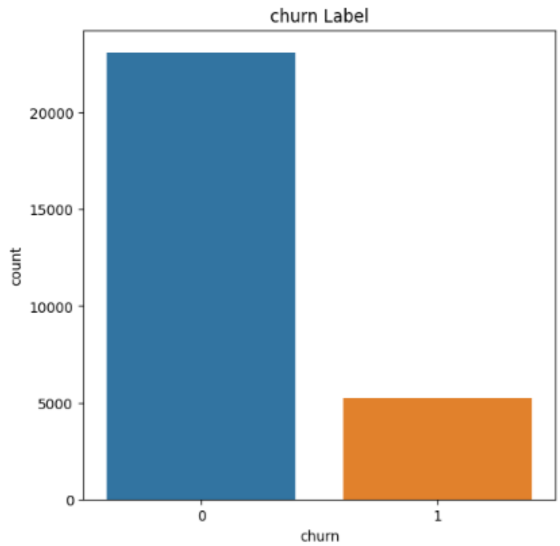
Imported the Churns Dataset

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns

        from sklearn.linear_model import LogisticRegression
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier
        from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score
        from sklearn.metrics import confusion_matrix, roc_auc_score, roc_curve, classification_report, precision_recall_curve
        from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold, GridSearchCV, RandomizedSearchCV

        from collections import Counter

In [2]: df = pd.read_csv("churn_prediction2.csv")
        df.shape

Out[2]: (28382, 11)

In [3]: df['churn'].value_counts()

Out[3]: 0    23122
        1     5260
        Name: churn, dtype: int64

In [4]: df.isnull().sum()

Out[4]: customer_id                0
        vintage                    0
        age                        0
        gender                   525
        dependents              2463
        occupation                80
        city                     803
        customer_nw_category       0
        branch_code                0
        churn                      0
```

We have used **SMOTE** (Synthetic Minority Oversampling Technique) in our dataset. This technique will duplicate the tuples of minority class and it will balance our Class Label.

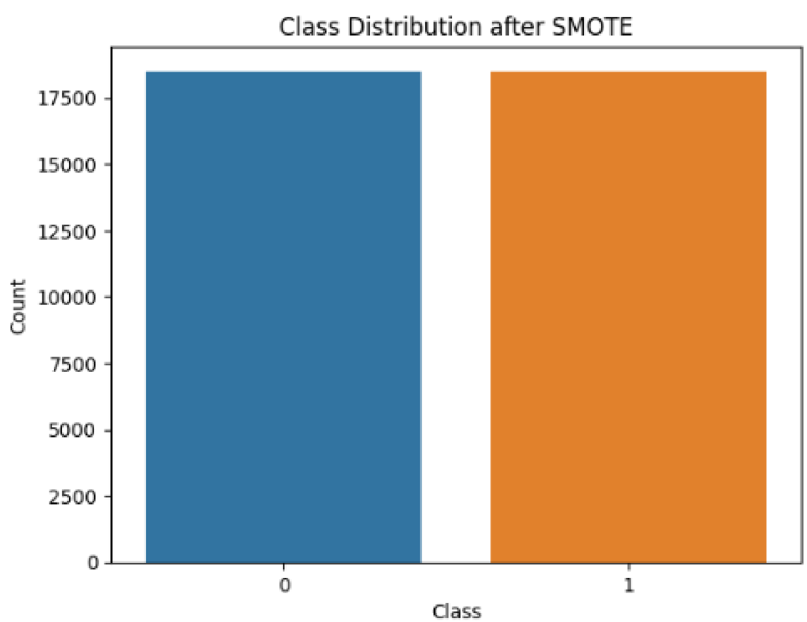
Before Applying the Smote technique the churn attributes was Unbalanced

```
In [8]: df.churn.value_counts()
        plt.figure(figsize=(6,6))
        sns.countplot(x='churn', data=df)
        plt.title('churn Label')
        plt.show()
```



After using SMOTE churn class label get balance :

```
In [18]: #smote technique
         from imblearn.over_sampling import SMOTE
         counter = Counter(y_train)
         print('Before',counter)
         # oversampling the train dataset using SMOTE
         smt = SMOTE()
         #X_train, y_train = smt.fit_resample(X_train, y_train)
         X_train_sm, y_train_sm = smt.fit_resample(X_train, y_train)
         counter = Counter(y_train_sm)
         print('After',counter)
         sns.countplot(x=y_train_sm)
         plt.title('Churn Label after SMOTE')
         plt.xlabel('Count')
         plt.ylabel('Churn')
         plt.show()

         Before Counter({0: 18497, 1: 4208})
         After Counter({0: 18497, 1: 18497})
```



5. Partition the dataset in training and testing:

In [51]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [52]:
```
df = pd.read_csv('churn3.csv')
```

In [53]:
```
df.head()
```

Out[53]:

	Unnamed: 0	customer_id	vintage	age	gender	dependents	occupation	city	customer_nw_category	branch_code	churn
0	0	1	2101	66	1.0	0.0	1.0	187.0	2	755	0
1	1	2	2348	35	1.0	0.0	1.0	NaN	2	3214	0
2	2	4	2194	31	1.0	0.0	0.0	146.0	2	41	0
3	3	5	2329	90	NaN	NaN	1.0	1020.0	2	582	1
4	4	6	1579	42	1.0	2.0	1.0	1494.0	3	388	1

In [54]:
```
# checking for missing values
df.isnull().sum()
```

Out[54]:
```
Unnamed: 0                 0
customer_id                0
vintage                    0
age                        0
gender                   525
dependents              2463
occupation                80
city                     803
customer_nw_category       0
branch_code                0
churn                      0
dtype: int64
```

Dividing the dataset into training and testing

In [55]:
```
# checking the distribution of Target Variable
df['churn'].value_counts()
```

Out[55]:
```
0    23122
1     5260
Name: churn, dtype: int64
```

In [56]:
```
X = df.drop(columns='churn', axis=1)
y = df['churn']
```

In [59]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.2, stratify=Y, random_state=2)
```

In [45]:
```
from sklearn.metrics import accuracy_score
```

In [46]:
```
#Train test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2)
```

In [60]:
```
# printing out train and test sets

print('X_train : ')
print(X_train.head())
print('')
print('X_test : ')
print(X_test.head())
print('')
print('y_train : ')
print(y_train.head())
print('')
print('y_test : ')
print(y_test.head())
```

```
X_train :
       Unnamed: 0  customer_id  vintage  age  gender  dependents  occupation  \
24249       24249        25876     1994   26     1.0         0.0         1.0
16252       16252        17341     2037   38     1.0         0.0         1.0
2968         2968         3165     2043   56     1.0         2.0         1.0
27254       27254        29100     1948   41     0.0         0.0         1.0
92             92           96     2424   29     0.0         0.0         0.0

         city  customer_nw_category  branch_code
24249  1540.0                     3         1301
16252   551.0                     3          550
2968     15.0                     2          237
27254  1096.0                     3          578
92      409.0                     2           26

X_test :
       Unnamed: 0  customer_id  vintage  age  gender  dependents  occupation  \
387           387          416     2133   47     1.0         0.0         1.0
17340       17340        18504     2269   41     1.0         2.0         0.0
5285         5285         5626     1781   42     0.0         0.0         0.0
2220         2220         2380     2212   54     1.0         0.0         0.0
12902       12902        13777     2085   40     0.0         0.0         1.0

         city  customer_nw_category  branch_code
387     363.0                     2           31
17340   395.0                     2         2212
5285    146.0                     2          312
2220   1181.0                     3          235
12902  1214.0                     3          787
```

6. T-Test :

In [13]:
```
#t-test for charges
stats.ttest_1samp(data, 0)
```

Out[13]:
```
TtestResult(statistic=177.91347020427807, pvalue=0.0, df=4718)
```

In [12]:
```
#t-test for charges
stats.ttest_1samp(data, 1)
```

Out[12]:
```
TtestResult(statistic=91.77684704905205, pvalue=0.0, df=4718)
```

In [14]:
```
t_statistic, p_value = stats.ttest_1samp(a=data, popmean=5000)
print(t_statistic , p_value)
```

```
-430505.20230592595 0.0
```

In [ ]:

In [ ]:

7. One Hot Encoding for all String values in our Area Attribute:

Before:

In [32]:
```
data = pd.read_csv("House_Rent_main6.csv")
data
```

Out[32]:

	BHK	Rent	Size	Area Type	Area Locality	City	Furnishing Status	Tenant Preferred	Bathroom
0	2	10000	1100	1	Bandel	1	1	Bachelors/Family	2
1	2	20000	800	1	Phool Bagan, Kankurgachi	1	2	Bachelors/Family	1
2	2	17000	1000	1	Salt Lake City Sector 2	1	2	Bachelors/Family	1
3	2	10000	800	1	Dumdum Park	1	1	Bachelors/Family	1
4	2	7500	850	2	South Dum Dum	1	1	Bachelors	1

After:

In [33]:
```
# Load your data into a pandas DataFrame
df = pd.read_csv('House_Rent_main6.csv')

# Identify the categorical variable(s) you want to encode
cat_cols = ['Area Locality']

# Perform one-hot encoding using pandas' get_dummies() function
df_encoded = pd.get_dummies(df, columns=cat_cols)

df_encoded
```

Out[33]:

	BHK	Rent	Size	Area Type	City	Furnishing Status	Tenant Preferred	Bathroom	Area Locality_ Beeramguda, Ramachandra Puram, NH 9	Area Locality_ in Boduppal, NH 2 2	...	Area Locality_sra	Area Locality_sri sai arcade madinaguda	Area Locality_sspdl Mayfair	Locality_s
0	2	10000	1100	1	1	1	Bachelors/Family	2	0	0	...	0	0	0	
1	2	20000	800	1	1	2	Bachelors/Family	1	0	0	...	0	0	0	
2	2	17000	1000	1	1	2	Bachelors/Family	1	0	0	...	0	0	0	
3	2	10000	800	1	1	1	Bachelors/Family	1	0	0	...	0	0	0	
4	2	7500	850	2	1	1	Bachelors	1	0	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
4714	2	15000	1000	2	6	2	Bachelors/Family	2	0	0	...	0	0	0	
4715	3	29000	2000	1	6	2	Bachelors/Family	3	0	0	...	0	0	0	
4716	3	35000	1750	2	6	2	Bachelors/Family	3	0	0	...	0	0	0	
4717	3	45000	1500	2	6	2	Family	2	0	0	...	0	0	0	
4718	2	15000	1000	2	6	1	Bachelors	2	0	0	...	0	0	0	

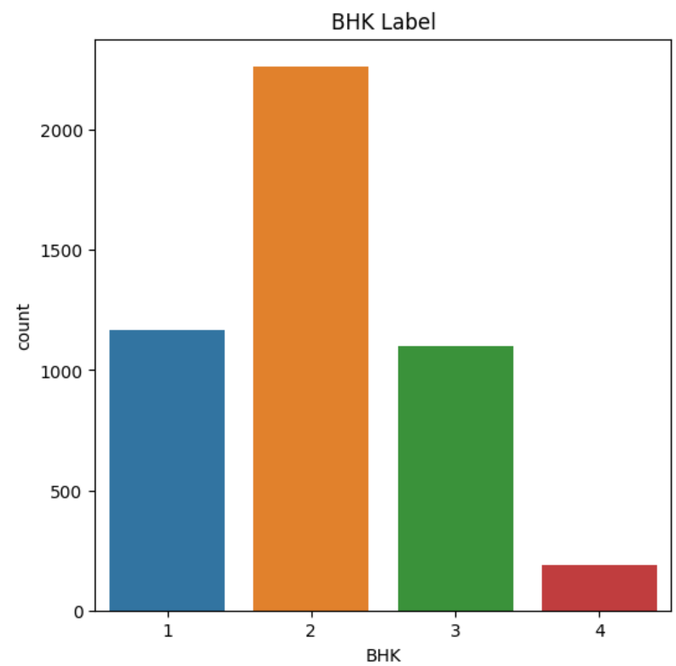
4719 rows × 2234 columns

8. Applying Classification Modelling:

We will use a variety of classification methods to determine which algorithms are most accurate and suitable for our dataset.

There are 4 class labels in our dataset.

```
In [8]: df.BHK.value_counts()
        plt.figure(figsize=(6,6))
        sns.countplot(x='BHK', data=new_df)
        plt.title('BHK Label')
        plt.show()
```



Splitting into training and testing dataset

```
In [3]: #import pandas
        import pandas as pd
        #import numpy
        import numpy as np
        #import matplotlib
        import matplotlib.pyplot as plt
        #import seaborn
        import seaborn as sns
```

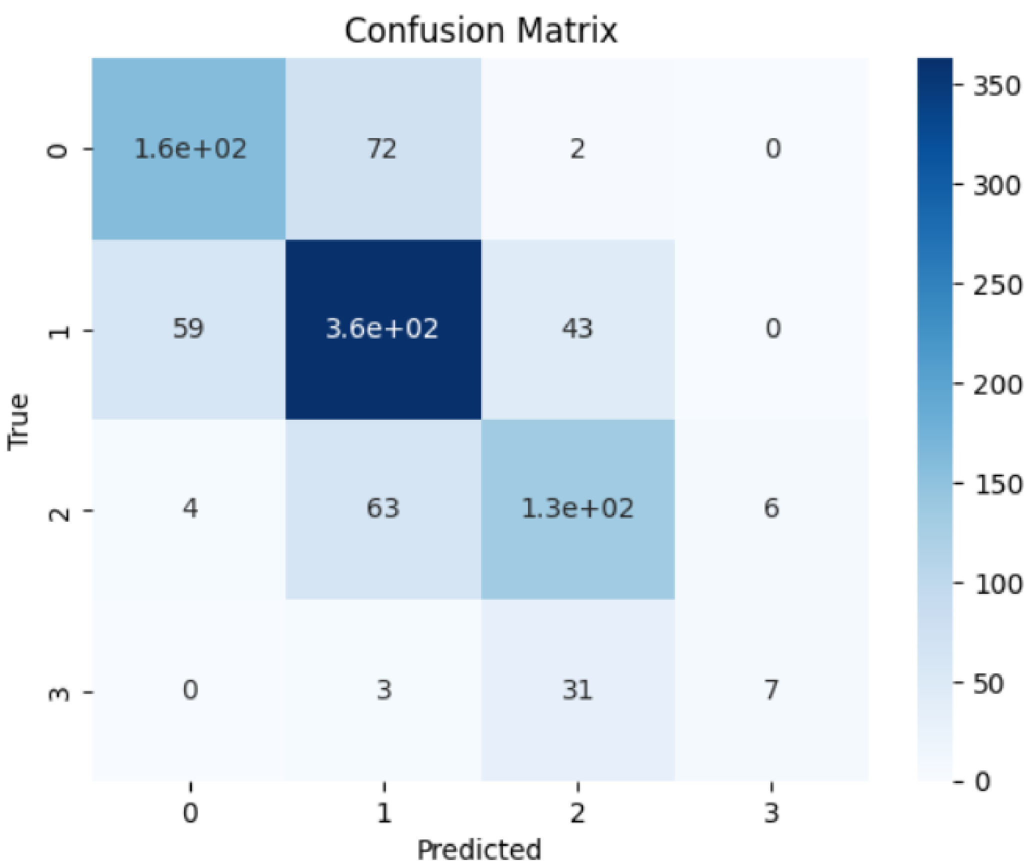
```
In [9]: from sklearn.datasets import make_classification
        X, y = make_classification(n_classes=2, class_sep=0.5,
        weights=[0.05, 0.95], n_informative=2, n_redundant=0, flip_y=0,
        n_features=2, n_clusters_per_class=1, n_samples=1000, random_state=10)
```

```
In [10]: from sklearn.model_selection import train_test_split
         # split into 75:25 ratio
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

8.a) Applying KNN Classifier Algorithm:

In [107]:
```
#KNN classifer
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
#Accuracy and Confusion matrix
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(accuracy_score(y_test, model.predict(X_test)))

# Plot the confusion matrix using heatmap
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

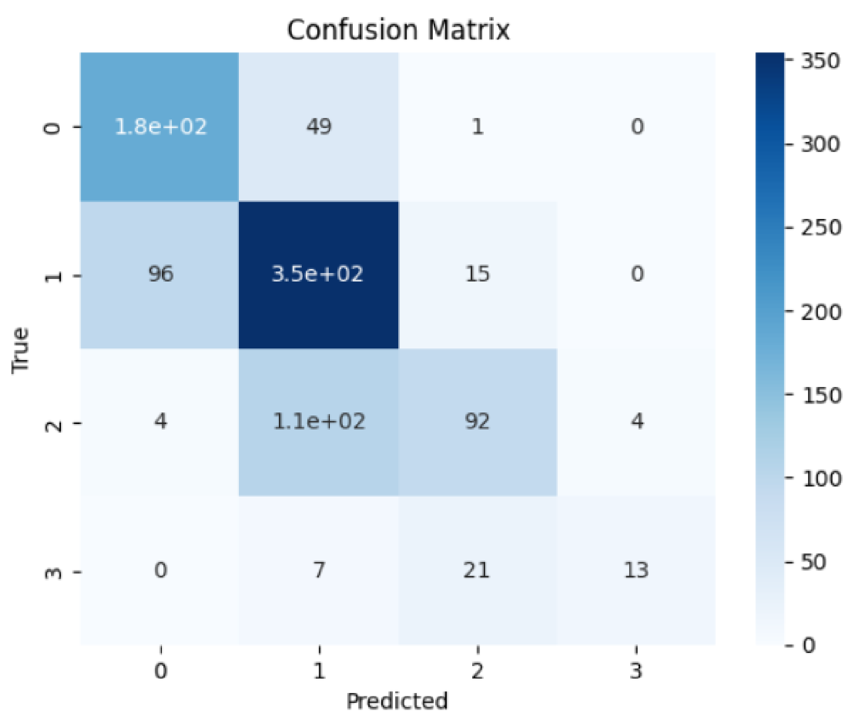
```
0.7002118644067796
```



8.b) Applying Naive Bayes Classifier Algorithm:

In [66]:
```
#Naive Bayes Classifier

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(df.drop('BHK', axis=1), df['BHK'], test_size=0.2, random_state=42)

# Convert categorical variables into numerical variables
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)

# Train the Gaussian Naive Bayes model
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Test the model and calculate accuracy
y_pred = gnb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))

cm = confusion_matrix(y_test, y_pred)
print(cm)

# Plot the confusion matrix using heatmap
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

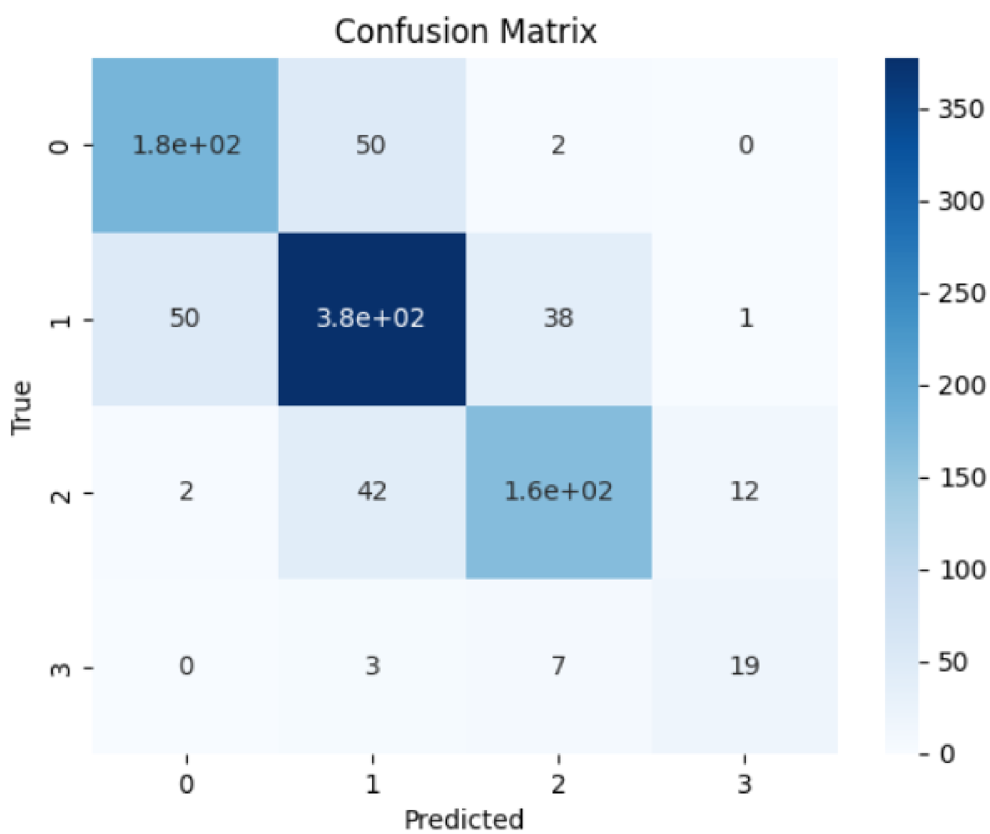
```
Accuracy: 0.68
[[182  49   1   0]
 [ 96 354  15   0]
 [  4 106  92   4]
 [  0   7  21  13]]
```



8.c) Applying Decision Tree Classifier Algorithm:

In [81]:
```
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

from sklearn import metrics #Import scikit-learn metrics module for accuracy calculation
cm = confusion_matrix(y_test, y_pred)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))

# Plot the confusion matrix using heatmap
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

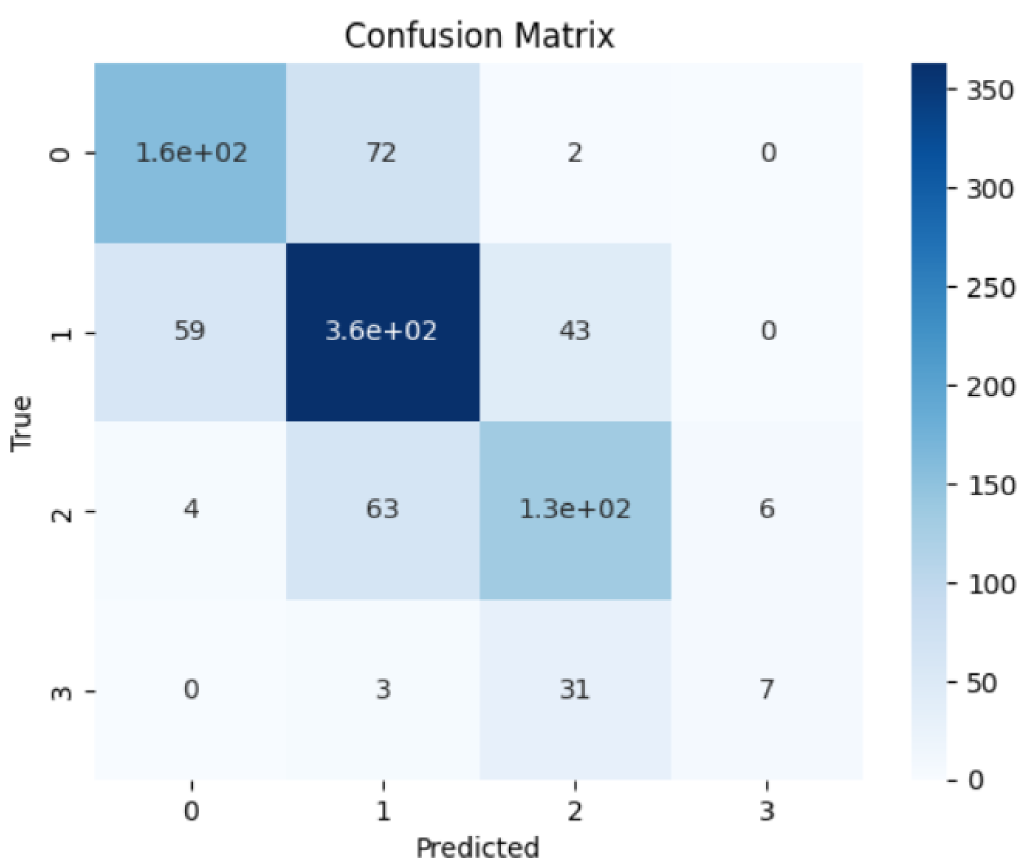
```
Accuracy: 0.7807203389830508
```



8.d) Applying Random Forest Classifier Algorithm:

In [108]:
```
#Random forest classifier
X_train, X_test, y_train, y_test = train_test_split(df.drop('BHK', axis=1), df['BHK'], test_size=0.2, random_state=42)

# Scale the features using standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Train the Random Forest Classifier
rfc = RandomForestClassifier(n_estimators=100, random_state=42)
rfc.fit(X_train, y_train)

# Test the Random Forest Classifier
y_pred = rfc.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy of Random Forest Classifier: {accuracy:.2f}")

# Plot the confusion matrix using heatmap
sns.heatmap(cm, annot=True, cmap="Blues")
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.show()
```

```
Accuracy of Random Forest Classifier: 0.83
```



Conclusion:

With the above experiments we can conclude that Random Forest Classifier gives us the most accuracy.
Confusion matrix:
True Positive (TP) = 160
True Negative (TN) = 7
False Positive (FP) = 74
False Negative (FN) = 63
Accuracy Rate: 0.83 or 83%

In conclusion, we have performed classification operations on the house rent dataset using various algorithms such as KNN Classifier, Decision Tree, Random Forest and Naive Bayes Classifier. We also evaluated the performance of these algorithms using metrics such as accuracy and confusion Matrix.
Based on our experiments, we found that the **Random Forest Classifier** performed the best with an accuracy of around 83%, which is a reasonably good performance given the complexity and variability of the data. The other algorithms also performed reasonably well, with accuracy ranging from 68% to 78%. Overall, the results suggest that machine learning algorithms can be useful in predicting the house rent prices based on various features such as location, area, number of rooms, etc. However, there is still room for improvement, and more sophisticated techniques can be explored to improve the accuracy and performance of the models.