ClassificationChurnLatestFeb26

March 3, 2025

0.1 CLASSIFICATION ANALYSIS

DATASET: CUSTOMER CHURN FROM Kaggle.com

[3]:
```
# Necessary imports for preprocessing and machine learning

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, MinMaxScaler
# if you want to use one Hot encoder then you need to import the below onw
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer   # interpolation for missing values
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import RocCurveDisplay, confusion_matrix, accuracy_score,␣
 ↪precision_score, recall_score, f1_score, classification_report
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
# load the dataset

df=pd.read_csv(r"Customer_Churn_Dataset.csv")
```

[5]:
```
df.head(5)
```

[5]:

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	\
0	1	56.0	38	112.48	61.87	
1	2	69.0	28	174.51	310.65	
2	3	46.0	48	35.11	310.02	
3	4	32.0	38	123.14	434.93	
4	5	60.0	31	95.38	286.54	

	Support_Calls	Contract_Type	Payment_Method	Has_Additional_Services	\
0	8	Two Year	Debit Card	1	

```
1              6        Two Year     Debit Card                      1
2              1  Month-to-Month            UPI                      1
3              7        One Year         PayPal                      0
4              1  Month-to-Month    Credit Card                      0

   Churn
0      0
1      0
2      0
3      0
4      1
```

[6]:
```
df.dtypes
```

[6]:
```
CustomerID                    int64
Age                         float64
Subscription_Length_Months    int64
Monthly_Bill                float64
Total_Usage_GB              float64
Support_Calls                 int64
Contract_Type                object
Payment_Method               object
Has_Additional_Services       int64
Churn                         int64
dtype: object
```

[ ]:
```

```

Descriptive Statistics of Data

[8]:
```
df.describe().T
```

[8]:
```
                             count         mean          std    min  \
CustomerID                 10000.0  5000.500000  2886.895680   1.00
Age                         9998.0    46.181136    16.551075   2.00
Subscription_Length_Months 10000.0    29.901700    16.996680   1.00
Monthly_Bill               10000.0   104.676996    54.670610  10.00
Total_Usage_GB             10000.0   252.664805   145.052928   0.51
Support_Calls              10000.0     4.538300     2.876658   0.00
Has_Additional_Services    10000.0     0.506500     0.499983   0.00
Churn                      10000.0     0.210700     0.407826   0.00

                                25%      50%        75%       max
CustomerID                2500.750  5000.50  7500.2500  10000.00
Age                         32.000    46.00    61.0000    250.00
Subscription_Length_Months  15.000    29.00    45.0000     59.00
Monthly_Bill                57.575   104.54   151.2475    199.98
Total_Usage_GB             129.045   253.18   377.5475   2000.11
```

```
Support_Calls                  2.000     5.00     7.0000      9.00
Has_Additional_Services        0.000     1.00     1.0000      1.00
Churn                          0.000     0.00     0.0000      1.00
```

[9]:
```
df.describe(include=['object']).T
```

[9]:
```
                 count unique             top  freq
Contract_Type     9998      3  Month-to-Month  3352
Payment_Method   10000      4      Debit Card  2540
```

[ ]:
```

```

Checking for Null Values

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   CustomerID                  10000 non-null  int64
 1   Age                         9998 non-null   float64
 2   Subscription_Length_Months  10000 non-null  int64
 3   Monthly_Bill                10000 non-null  float64
 4   Total_Usage_GB              10000 non-null  float64
 5   Support_Calls               10000 non-null  int64
 6   Contract_Type               9998 non-null   object
 7   Payment_Method              10000 non-null  object
 8   Has_Additional_Services     10000 non-null  int64
 9   Churn                       10000 non-null  int64
dtypes: float64(3), int64(5), object(2)
memory usage: 781.4+ KB
```

[12]:
```
# find the total number of missing values for each variable

df.isnull().sum()
```

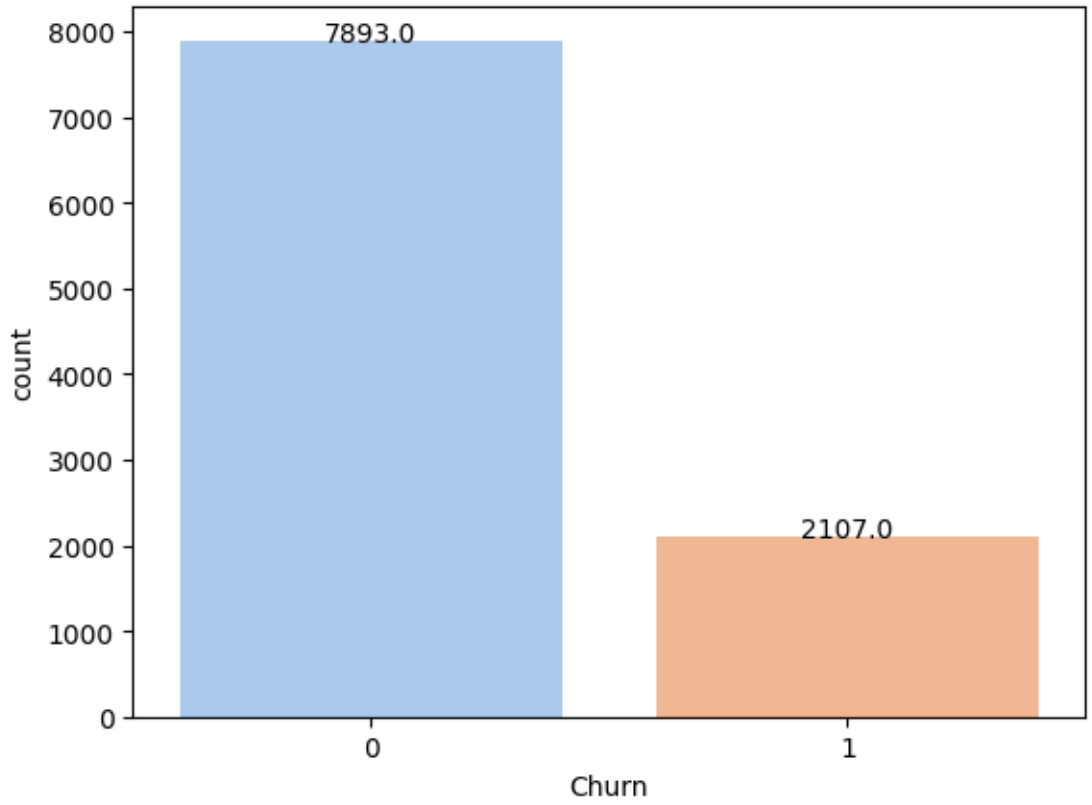
[12]:
```
CustomerID                  0
Age                         2
Subscription_Length_Months  0
Monthly_Bill                0
Total_Usage_GB              0
Support_Calls               0
Contract_Type               2
Payment_Method              0
Has_Additional_Services     0
Churn                       0
```

```
dtype: int64
```

[ ]:

Is the dataset balanced?

[14]:
```
# Create the countplot
#ax = sns.countplot(data=df, x='Churn')
ax = sns.countplot(data=df, x='Churn', hue='Churn', palette='pastel',␣
 ↪legend=False)
# Add count labels on top of the bars
for p in ax.patches:
    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.
 ↪get_height()),
                ha='center', va='baseline')

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



The dataset is not balanced. We balance it using SMOTE

[16]:
```
# count the number of people who have customer churn yes against number of␣
 ↪people who have customer churn no to see if the label/response
# is balanced or not
#customer churn yes = 1

# Number of people with customer churn yes
(df[['Churn']]==1).sum()      # you can also use      #df.Churn.
 ↪value_counts()[1]
```

[16]:
```
Churn    2107
dtype: int64
```

[17]:
```
# Number of people with customer churn no
#Customer churn no = 0
(df[['Churn']]==0).sum()  # you can also use  #  df.Churn.value_counts()[0]
```

[17]:
```
Churn    7893
dtype: int64
```

[18]:
```
# calculate probability of customer churn yes

(df[['Churn']]==1).sum() / len(df['Churn'])
```

[18]:
```
Churn    0.2107
dtype: float64
```

[19]:
```
# calculate probability of customer churn no

(df[['Churn']]==0).sum() / len(df['Churn'])
```

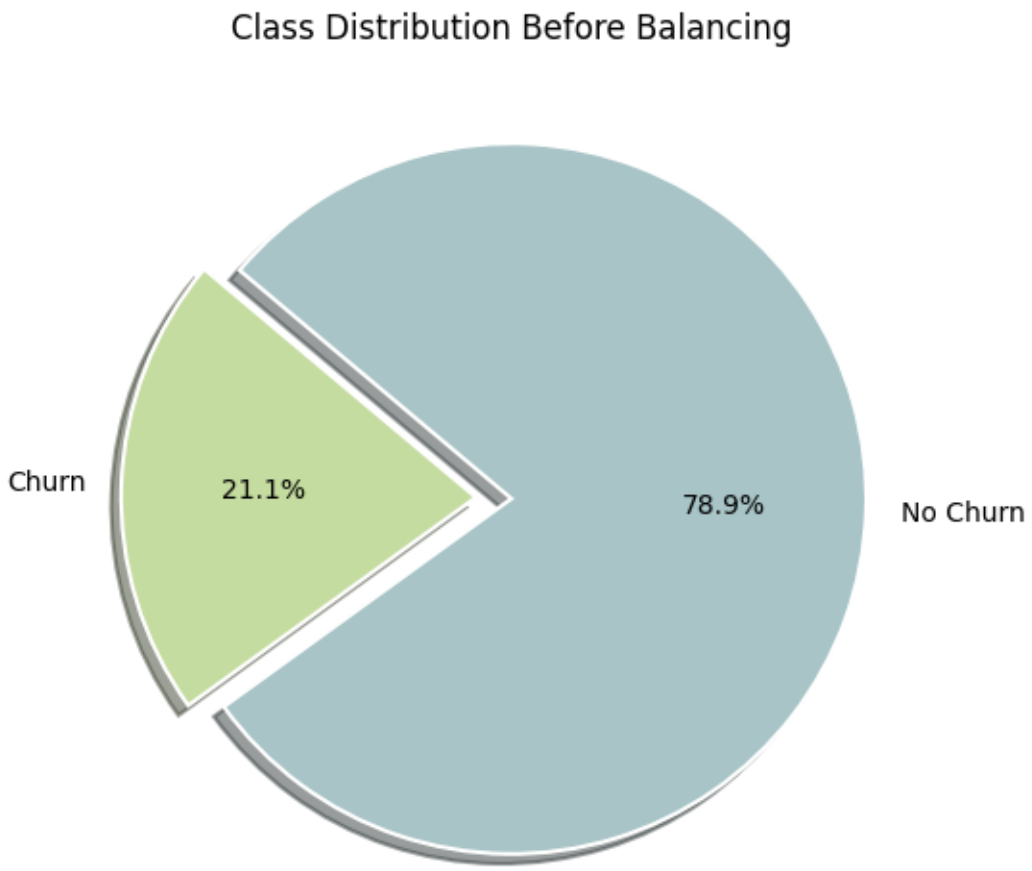
[19]:
```
Churn    0.7893
dtype: float64
```

[ ]:
```

```

Pie Chart to show data imbalance

[21]:
```
l = ['Churn', 'No Churn']
s = [ (df['Churn']==1).sum(), (df['Churn']==0).sum()]
c = ['#C5DCA0', '#A8C4C7']
```

[22]:
```
# Plot pie chart
plt.figure(figsize=(6,6))
plt.pie(s, labels=l, colors=c, explode=(0.1, 0), shadow=True, autopct='%1.1f%%',
        startangle=140, wedgeprops={'edgecolor': 'white', 'linewidth': 1.5})

plt.title("Class Distribution Before Balancing")
plt.show()
```



[ ]:

Handling Missing values

[24]:
```
# How to get the list of all columns in my dataset

columns_list=df.columns.tolist()
```

[25]:
```
columns_list
```

[25]:
```
['CustomerID',
 'Age',
 'Subscription_Length_Months',
 'Monthly_Bill',
 'Total_Usage_GB',
 'Support_Calls',
 'Contract_Type',
 'Payment_Method',
```

```
 'Has_Additional_Services',
 'Churn']
```

[26]:
```
# We need to create a variable that stores the name of categorical columns
# as well a varibale that stores the name of numerical variable.
# because we need to handle missing values for categorical using mode technique
# we also need to replace the missing values of numerical variable using mean.␣
 ↪Can also use linear interpocation

# List of categorical variables (dtype == object or category)
categorical_cols = df.select_dtypes(include=['object', 'category']).columns.
 ↪tolist()

# List of numerical variables (dtype == int or float)
numerical_cols = df.select_dtypes(include=['number']).columns.tolist()

# Display the lists
print("Categorical columns:", categorical_cols)
print("Numerical columns:", numerical_cols)
```

```
Categorical columns: ['Contract_Type', 'Payment_Method']
Numerical columns: ['CustomerID', 'Age', 'Subscription_Length_Months',
'Monthly_Bill', 'Total_Usage_GB', 'Support_Calls', 'Has_Additional_Services',
'Churn']
```

[27]:
```
# Input categorical variable with mode (replace missing values of a categorical␣
 ↪variable with mode)

for col in categorical_cols:
    df[col] = df[col].fillna(df[col].mode()[0])
    #for col in categorical_cols:
    #df[col].fillna(df[col].mode()[0], inplace=True)

# note that we may have multiple values for mode. mode()[0] returns the first␣
 ↪most frequent category.
# mode()[1] returns the second mode. This is because the mode() function can␣
 ↪potentially return multiple values if there
# is more than one mode (i.e., multiple values that occur with the same highest␣
 ↪frequency).
# If you don't use [0] and the mode() returns more than one value, the .
 ↪fillna() function will not know which value to use
# If you remove it, you might get an error if there is more than one mode,
# because mode() can return multiple values and .fillna() requires only one␣
 ↪value.

# If you remove inplace=True, the original DataFrame remains unchanged unless␣
 ↪you explicitly reassign the result.
```

[28]:
```
df.isnull().sum()  # all missing values under categorical variables have been␣
↪replaced with their mode
```

[28]:
```
CustomerID                  0
Age                         2
Subscription_Length_Months  0
Monthly_Bill                0
Total_Usage_GB              0
Support_Calls               0
Contract_Type               0
Payment_Method              0
Has_Additional_Services     0
Churn                       0
dtype: int64
```

[29]:
```
# handling missing values of numerical variables
# Input numerical column (variable), which is Age, with mean
df['Age'] = df['Age'].fillna(df['Age'].mean())
#df['age'].fillna(df['age'].mean(), inplace=True)
```

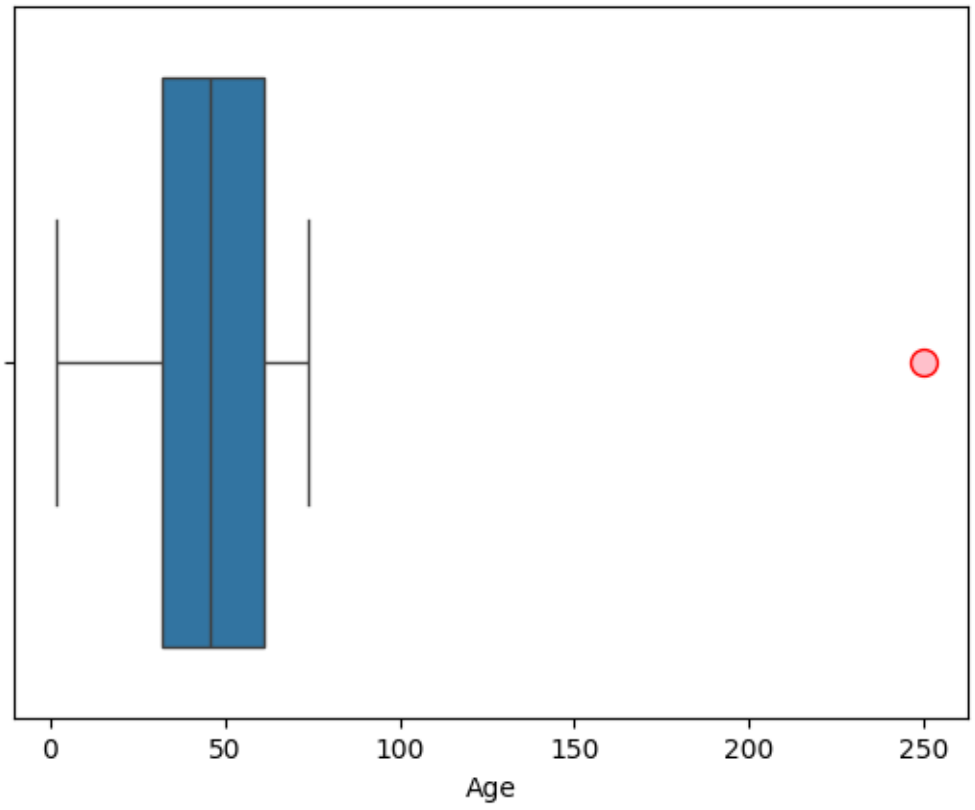
[30]:
```
df.isnull().sum()   # zero missing values for numerical variable (age)
```

[30]:
```
CustomerID                  0
Age                         0
Subscription_Length_Months  0
Monthly_Bill                0
Total_Usage_GB              0
Support_Calls               0
Contract_Type               0
Payment_Method              0
Has_Additional_Services     0
Churn                       0
dtype: int64
```

[ ]:
```

```

Find and remove outliers from numerical variable

[32]:
```
# Create a boxplot for the 'age' column to detect outliers
#sns.boxplot(data=df, x='Age');
# Customizing outlier appearance
sns.boxplot(data=df, x='Age',
            flierprops={"marker": "o", "markerfacecolor": "pink",␣
↪"markeredgecolor": "red", "markersize": 10});
```

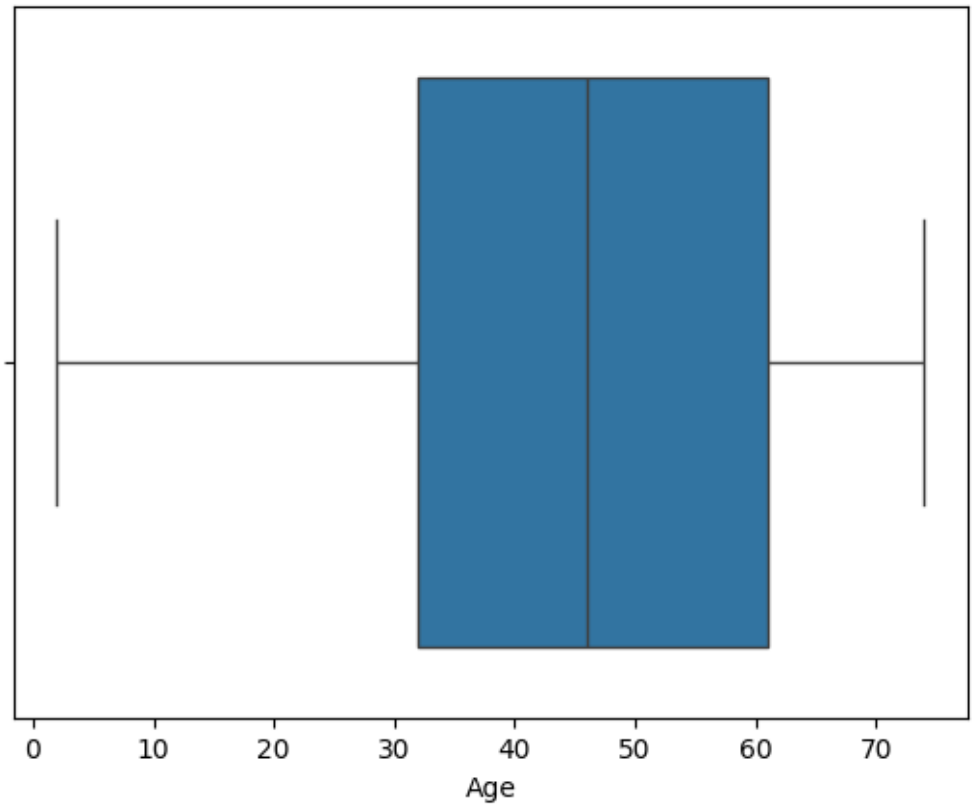


[33]:
```
# Calculate the IQR for the 'age' column
Q1 = df['Age'].quantile(0.25)  # 25th percentile
Q3 = df['Age'].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1  # Interquartile range

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out the outliers
df1 = df[(df['Age'] >= lower_bound) & (df['Age'] <= upper_bound)]

# df1 is a new datafram after removing outliers

# Display the results (rows without outliers)
print("Cleaned DataFrame (Without Age Outliers):")
print(df1)
```

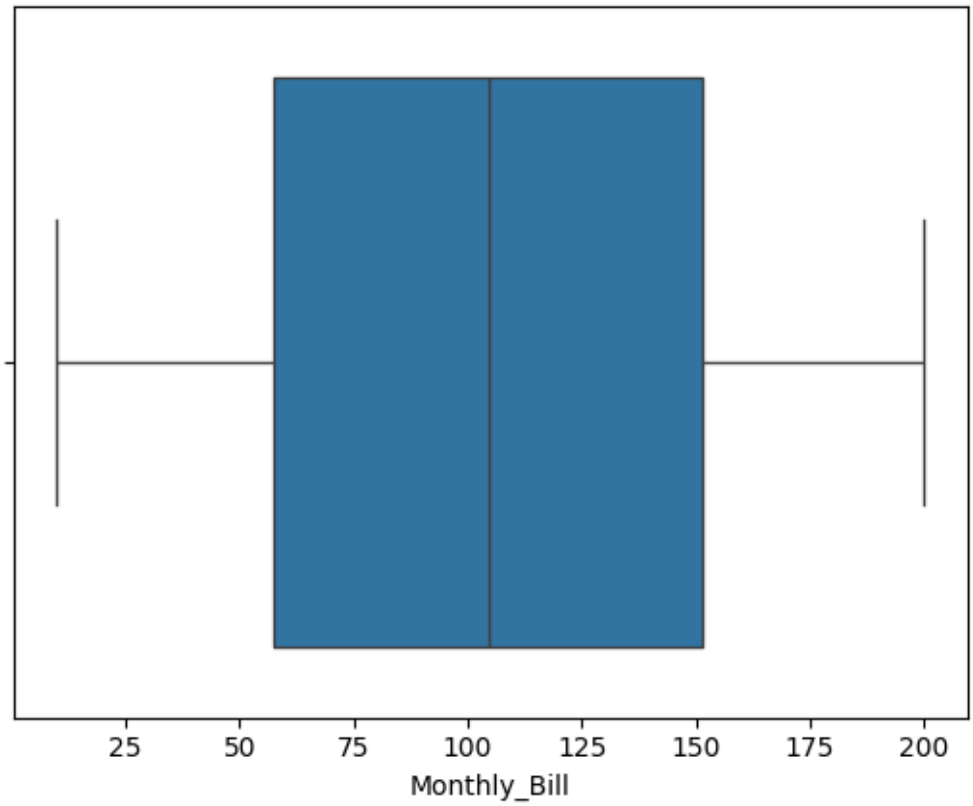
```
Cleaned DataFrame (Without Age Outliers):
      CustomerID   Age  Subscription_Length_Months  Monthly_Bill  \
0              1  56.0                          38        112.48
```

```
1              2  69.0                      28        174.51
2              3  46.0                      48         35.11
3              4  32.0                      38        123.14
4              5  60.0                      31         95.38
…            …   …                       …           …
9995        9996  40.0                       8        173.06
9996        9997  58.0                      33        175.46
9997        9998  57.0                      18         13.80
9998        9999  46.0                      45         20.59
9999       10000  33.0                      29        184.48

      Total_Usage_GB  Support_Calls   Contract_Type Payment_Method  \
0              61.87              8       Two Year     Debit Card
1             310.65              6       Two Year     Debit Card
2             310.02              1  Month-to-Month            UPI
3             434.93              7       One Year         PayPal
4             286.54              1  Month-to-Month    Credit Card
…               …               …              …           …
9995          127.42              2       Two Year    Credit Card
9996          385.53              1       Two Year     Debit Card
9997          115.07              3  Month-to-Month            UPI
9998          260.86              7       One Year         PayPal
9999           55.79              4  Month-to-Month            UPI

      Has_Additional_Services  Churn
0                           1      0
1                           1      0
2                           1      0
3                           0      0
4                           0      1
…                         …    …
9995                        1      0
9996                        0      0
9997                        1      0
9998                        1      0
9999                        1      0

[9999 rows x 10 columns]
```

[34]:
```
sns.boxplot(data=df1, x='Age');  # outliers have been removed
```



[35]:
```
# Find and remove outlier from numerical variable

# Create a boxplot for the 'Monthly_Bill' column to detect outliers
sns.boxplot(data=df, x='Monthly_Bill');
```

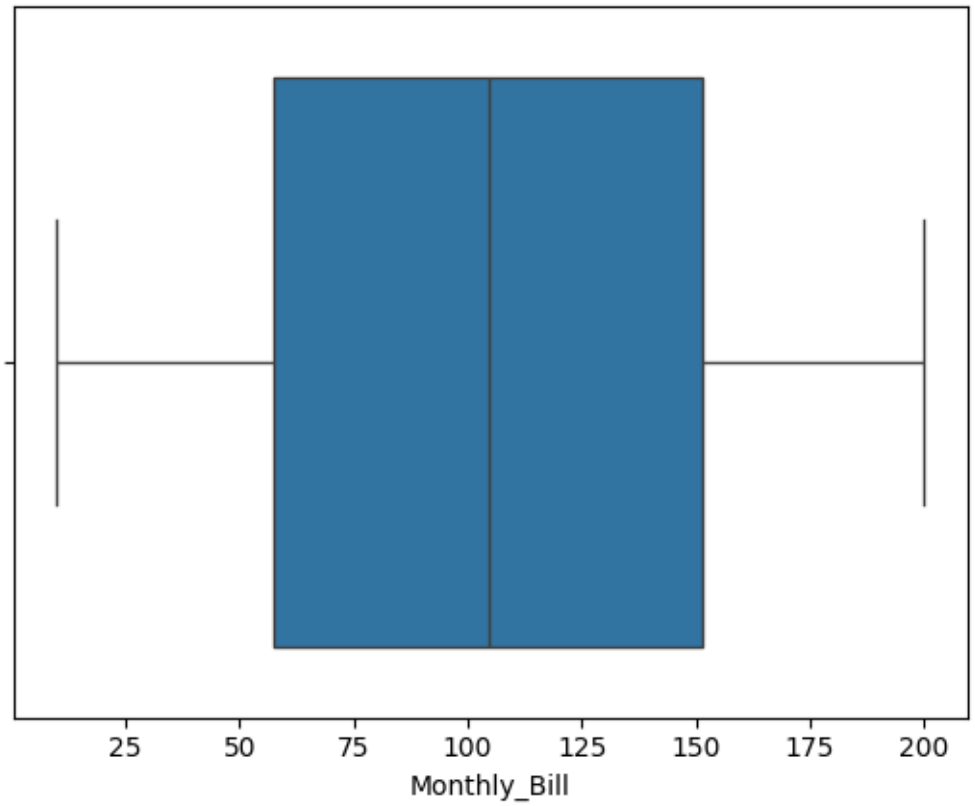


[36]:
```
# Calculate the IQR for the 'age' column
Q1 = df1['Monthly_Bill'].quantile(0.25)  # 25th percentile
Q3 = df1['Monthly_Bill'].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1  # Interquartile range

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out the outliers
df2 = df1[(df1['Monthly_Bill'] >= lower_bound) & (df1['Monthly_Bill'] <=␣
 ↪upper_bound)]

# df1 is a new dataframe after removing Age outliers and df2 is the new␣
 ↪dataframe after removing Age and MonthlyBill outliers

# Display the results (rows without outliers)
print("Cleaned DataFrame (Without Monthly_Bill Outliers):")
print(df2)
```

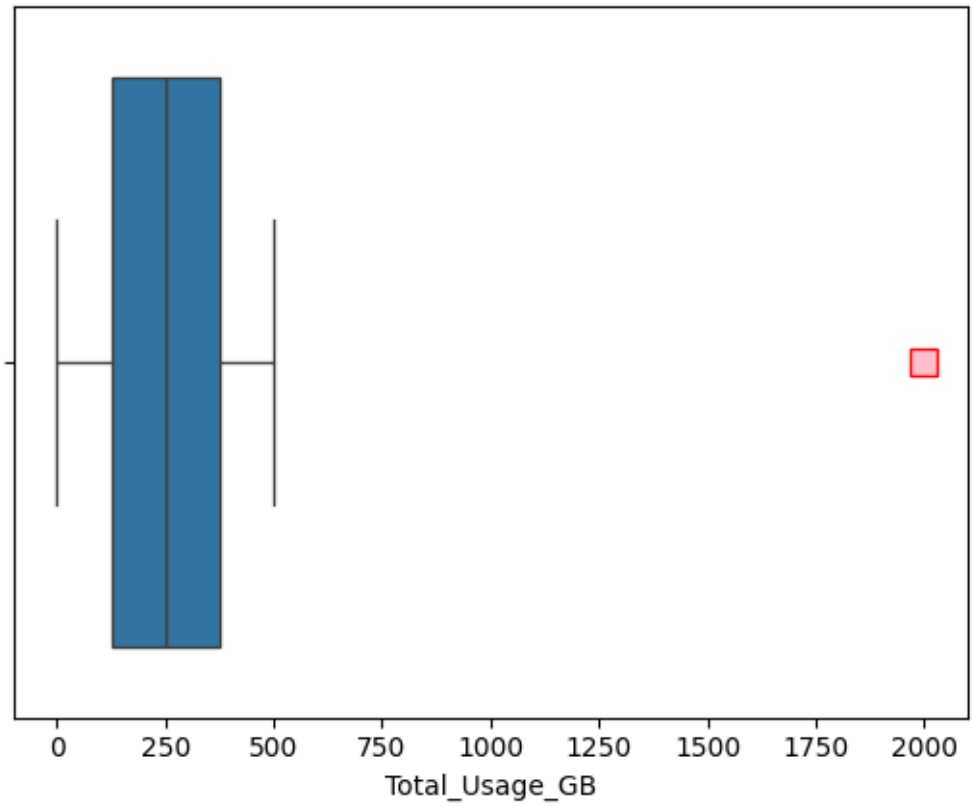
```
Cleaned DataFrame (Without Monthly_Bill Outliers):
```

```
      CustomerID   Age  Subscription_Length_Months  Monthly_Bill  \
0              1  56.0                          38        112.48
1              2  69.0                          28        174.51
2              3  46.0                          48         35.11
3              4  32.0                          38        123.14
4              5  60.0                          31         95.38
…            …   …                             …           …
9995        9996  40.0                           8        173.06
9996        9997  58.0                          33        175.46
9997        9998  57.0                          18         13.80
9998        9999  46.0                          45         20.59
9999       10000  33.0                          29        184.48

      Total_Usage_GB  Support_Calls   Contract_Type Payment_Method  \
0              61.87              8       Two Year     Debit Card
1             310.65              6       Two Year     Debit Card
2             310.02              1  Month-to-Month            UPI
3             434.93              7       One Year         PayPal
4             286.54              1  Month-to-Month    Credit Card
…               …               …              …           …
9995          127.42              2       Two Year    Credit Card
9996          385.53              1       Two Year     Debit Card
9997          115.07              3  Month-to-Month            UPI
9998          260.86              7       One Year         PayPal
9999           55.79              4  Month-to-Month            UPI

      Has_Additional_Services  Churn
0                           1      0
1                           1      0
2                           1      0
3                           0      0
4                           0      1
…                         …    …
9995                        1      0
9996                        0      0
9997                        1      0
9998                        1      0
9999                        1      0

[9999 rows x 10 columns]
```

[37]:
```
sns.boxplot(data=df2, x='Monthly_Bill');  # outliers have been removed
```



[38]:
```
#sns.boxplot(data=df2, x='Total_Usage_GB');  # with outliers
# Customizing outlier appearance
sns.boxplot(data=df2, x='Total_Usage_GB',
            flierprops={"marker": "s", "markerfacecolor": "pink",␣
 ↪"markeredgecolor": "red", "markersize": 10});
```

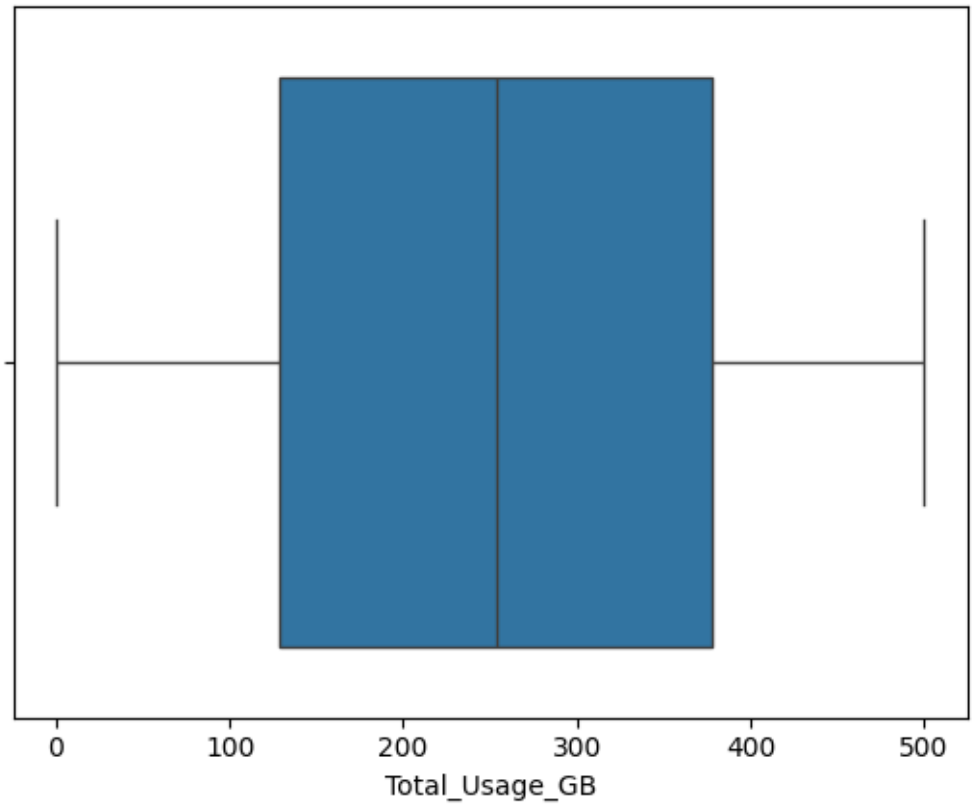


[39]:
```
# Calculate the IQR for the 'age' column
Q1 = df2['Total_Usage_GB'].quantile(0.25)  # 25th percentile
Q3 = df2['Total_Usage_GB'].quantile(0.75)  # 75th percentile
IQR = Q3 - Q1  # Interquartile range

# Define the lower and upper bounds for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter out the outliers
df3 = df2[(df2['Total_Usage_GB'] >= lower_bound) & (df2['Total_Usage_GB'] <=␣
 ↪upper_bound)]

# df3 is a new dataframe after removing Total Usage outliers.

# Display the results (rows without outliers)
print("Cleaned DataFrame (Without Total_Usage_GB Outliers):")
print(df3)
```

```
Cleaned DataFrame (Without Total_Usage_GB Outliers):
      CustomerID   Age  Subscription_Length_Months  Monthly_Bill  \
```

```
0              1  56.0                        38        112.48
1              2  69.0                        28        174.51
2              3  46.0                        48         35.11
3              4  32.0                        38        123.14
4              5  60.0                        31         95.38
…            …    …                         …            …
9995        9996  40.0                         8        173.06
9996        9997  58.0                        33        175.46
9997        9998  57.0                        18         13.80
9998        9999  46.0                        45         20.59
9999       10000  33.0                        29        184.48

      Total_Usage_GB  Support_Calls   Contract_Type Payment_Method  \
0              61.87              8        Two Year     Debit Card
1             310.65              6        Two Year     Debit Card
2             310.02              1  Month-to-Month            UPI
3             434.93              7        One Year         PayPal
4             286.54              1  Month-to-Month    Credit Card
…                …              …              …              …
9995          127.42              2        Two Year    Credit Card
9996          385.53              1        Two Year     Debit Card
9997          115.07              3  Month-to-Month            UPI
9998          260.86              7        One Year         PayPal
9999           55.79              4  Month-to-Month            UPI

      Has_Additional_Services  Churn
0                           1      0
1                           1      0
2                           1      0
3                           0      0
4                           0      1
…                         …    …
9995                        1      0
9996                        0      0
9997                        1      0
9998                        1      0
9999                        1      0

[9998 rows x 10 columns]
```

[40]:
```
sns.boxplot(data=df3, x='Total_Usage_GB');  # outliers have been removed
```



[41]:
```
type(df3)
```

[41]:
```
pandas.core.frame.DataFrame
```

[ ]:
```

```

Label Encoding for Categorical Features

[43]:
```
# in this datase, we have 2 categorical features (e.g., contract_type(monthly,␣
 ↪year-to-year, ..), and a response/label variable)

# we use the following function to encode categorical variable

from sklearn.preprocessing import LabelEncoder

lb=LabelEncoder()

for col in categorical_cols:    # we define categorical_cols   in line [73]
    df3.loc[:,col]=lb.fit_transform(df3[col])   # note that df3 is new␣
 ↪dataframe without outliers
```

[44]:
```
df3.head(5)
```

[44]:

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	\
0	1	56.0	38	112.48	61.87	
1	2	69.0	28	174.51	310.65	
2	3	46.0	48	35.11	310.02	
3	4	32.0	38	123.14	434.93	
4	5	60.0	31	95.38	286.54	

	Support_Calls	Contract_Type	Payment_Method	Has_Additional_Services	Churn
0	8	2	1	1	0
1	6	2	1	1	0
2	1	0	3	1	0
3	7	1	2	0	0
4	1	0	0	0	1

[45]:
```
numerical_cols
```

[45]:
```
['CustomerID',
 'Age',
 'Subscription_Length_Months',
 'Monthly_Bill',
 'Total_Usage_GB',
 'Support_Calls',
 'Has_Additional_Services',
 'Churn']
```

[46]:
```
df3.columns
```

[46]:
```
Index(['CustomerID', 'Age', 'Subscription_Length_Months', 'Monthly_Bill',
       'Total_Usage_GB', 'Support_Calls', 'Contract_Type', 'Payment_Method',
       'Has_Additional_Services', 'Churn'],
      dtype='object')
```

[47]:
```
# Standardize the numerical variable (Age variable)

from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

df3 = df3.copy();
scale_cols = [x for x in numerical_cols if x != 'CustomerID']
df3.loc[:,scale_cols] = scaler.fit_transform(df3[scale_cols]);  # we define␣
 ↪list of numerical variable in line [22]

# df3 is the new dataframe after standardizing the numerical variable
```

```
/var/folders/9p/02j0tmhs5w35zmgt4_pc9src0000gn/T/ipykernel_4362/3217076613.py:9:
```

```
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '[0.63793103 0.46551724 0.81034483 …
0.29310345 0.75862069 0.48275862]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  df3.loc[:,scale_cols] = scaler.fit_transform(df3[scale_cols]);  # we define
list of numerical variable in line [22]
/var/folders/9p/02j0tmhs5w35zmgt4_pc9src0000gn/T/ipykernel_4362/3217076613.py:9:
FutureWarning: Setting an item of incompatible dtype is deprecated and will
raise in a future error of pandas. Value '[0.88888889 0.66666667 0.11111111 …
0.33333333 0.77777778 0.44444444]' has dtype incompatible with int64, please
explicitly cast to a compatible dtype first.
  df3.loc[:,scale_cols] = scaler.fit_transform(df3[scale_cols]);  # we define
list of numerical variable in line [22]
```

[48]:
```
df3
```

[48]:

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	\
0	1	0.750000	0.637931	0.539425	
1	2	0.930556	0.465517	0.865933	
2	3	0.611111	0.810345	0.132172	
3	4	0.416667	0.637931	0.595536	
4	5	0.805556	0.517241	0.449416	
…	…	…	…	…	
9995	9996	0.527778	0.120690	0.858301	
9996	9997	0.777778	0.551724	0.870934	
9997	9998	0.763889	0.293103	0.020002	
9998	9999	0.611111	0.758621	0.055743	
9999	10000	0.430556	0.482759	0.918412	

	Total_Usage_GB	Support_Calls	Contract_Type	Payment_Method	\
0	0.122848	0.888889	2	1	
1	0.620926	0.666667	2	1	
2	0.619664	0.111111	0	3	
3	0.869745	0.777778	1	2	
4	0.572656	0.111111	0	0	
…	…	…	…	…	
9995	0.254084	0.222222	2	0	
9996	0.770842	0.111111	2	1	
9997	0.229359	0.333333	0	3	
9998	0.521242	0.777778	1	2	
9999	0.110675	0.444444	0	3	

	Has_Additional_Services	Churn
0	1	0
1	1	0
2	1	0
3	0	0

```
4                              0       1
…                            …     …
9995                           1       0
9996                           0       0
9997                           1       0
9998                           1       0
9999                           1       0

[9998 rows x 10 columns]
```

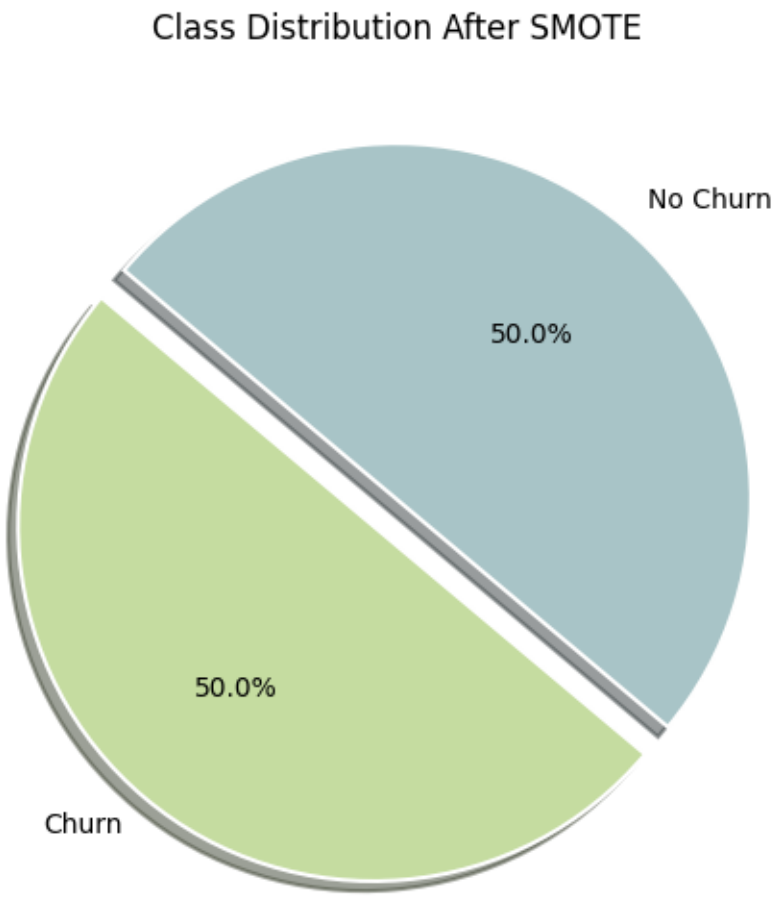
[ ]:

Define Features (X) and Response/Label (y)

[50]:
```
X= df3.drop(columns=['Churn'])
y=df3['Churn']
```

[51]:
```
# We saw earlier that the data is not balanced. We apply SMOTE sampling
 ↪technique to our imbalanced data

smote = SMOTE()

X_res, y_res = smote.fit_resample(X,y)
```

[52]:
```
from collections import Counter

# Count occurrences in resampled data
class_counts = Counter(y_res)  # Get counts of Churn (1) and No Churn (0)

l = ['Churn', 'No Churn']

# Values for pie chart
s = [class_counts[1], class_counts[0]]  # Count of churned and non-churned
 ↪customers

c = ['#C5DCA0', '#A8C4C7']

# Plot pie chart
plt.figure(figsize=(6,6))
plt.pie(s, labels=l, colors=c, explode=(0.1, 0), shadow=True, autopct='%1.1f%%',
        startangle=140, wedgeprops={'edgecolor': 'white', 'linewidth': 1.5})

plt.title("Class Distribution After SMOTE")
plt.show()
```



[53]:
```
y_res
```

[53]:
```
0        0
1        0
2        0
3        0
4        1
        ..
15779    1
15780    1
15781    1
15782    1
15783    1
Name: Churn, Length: 15784, dtype: int64
```

[54]:
```
y_res.value_counts()  # the response/ label has been balanced
```

[54]:
```
Churn
0    7892
1    7892
Name: count, dtype: int64
```

[55]:
```
# split into training and testing test

X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.
↪3, random_state=8)
```

[ ]:
```

```

Generating Models

[57]:
```
# Naive Bayes Model

nb_model = GaussianNB()
nb_model.fit(X_train, y_train)
y_pred_nb = nb_model.predict(X_test)
```

[58]:
```
# Decision Tree Model

dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)
y_pred_dt = dt_model.predict(X_test)
```

[59]:
```
# Random Forest Model

rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
y_pred_rf=rf_model.predict(X_test)
```

[60]:
```
# KNN model

knn_model =KNeighborsClassifier(n_neighbors=3)
# For n_neighbors =5, accuracy is 0.72
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
```

[ ]:
```

```

Evaluation: Confusion Matrix and Other Metrics

[62]:
```
def evaluate_model(y_test, y_pred, model_name):
    print(f"Model: {model_name}")
    print("Confusion Matrix:")
    print(confusion_matrix(y_test, y_pred))
    print(f"Accuracy:  {accuracy_score(y_test, y_pred):.2f}")
```

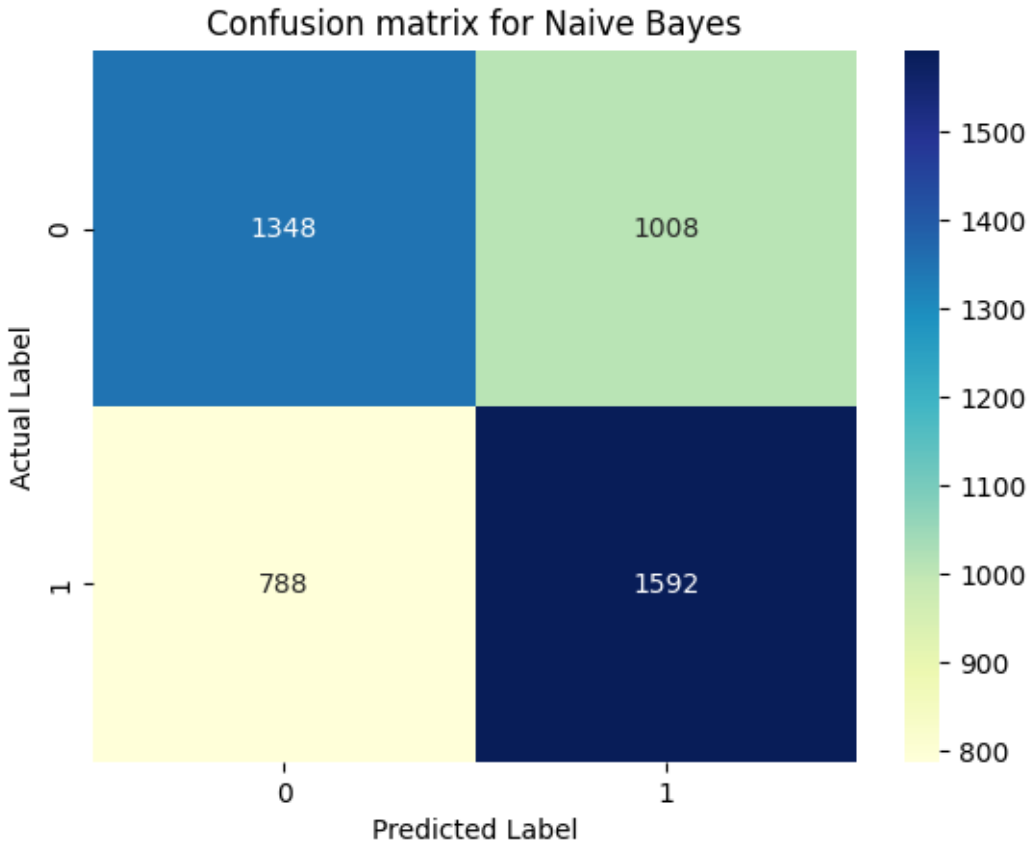
```
    print(f"Precision: {precision_score(y_test, y_pred):.2f}")
    print(f"PRecall:   {recall_score(y_test, y_pred):.2f}")
    print(f"F1 Score:  {f1_score(y_test, y_pred):.2f}\n")
```

[63]:
```
# Evaluate all models

evaluate_model(y_test, y_pred_nb, "Naive Bayes")
evaluate_model(y_test, y_pred_dt, "Decision Tree")
evaluate_model(y_test, y_pred_rf, "Random Forest")
evaluate_model(y_test, y_pred_knn, "KNN")
```

```
Model: Naive Bayes
Confusion Matrix:
[[1348 1008]
 [ 788 1592]]
Accuracy:  0.62
Precision: 0.61
PRecall:   0.67
F1 Score:  0.64

Model: Decision Tree
Confusion Matrix:
[[1763  593]
 [ 552 1828]]
Accuracy:  0.76
Precision: 0.76
PRecall:   0.77
F1 Score:  0.76

Model: Random Forest
Confusion Matrix:
[[2277   79]
 [ 676 1704]]
Accuracy:  0.84
Precision: 0.96
PRecall:   0.72
F1 Score:  0.82

Model: KNN
Confusion Matrix:
[[1282 1074]
 [ 593 1787]]
Accuracy:  0.65
Precision: 0.62
PRecall:   0.75
F1 Score:  0.68
```

[64]:
```
# plot confusion matrix using seaborn library

cm1=confusion_matrix(y_test, y_pred_nb)
cm2=confusion_matrix(y_test, y_pred_dt)
cm3=confusion_matrix(y_test, y_pred_rf)
cm4=confusion_matrix(y_test, y_pred_knn)
```

[65]:
```
# confusion matrix for Naive Bayes

sns.heatmap(cm1, annot=True, cmap='YlGnBu', fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.title('Confusion matrix for Naive Bayes');
```



Does overfitting happen for Naive Bayes? We need to compare the accuracy of testing against accuracy of training

[67]:
```
print ("Accuracy of Naive Bayes for training database is:", nb_model.
  ↪score(X_train, y_train))
```

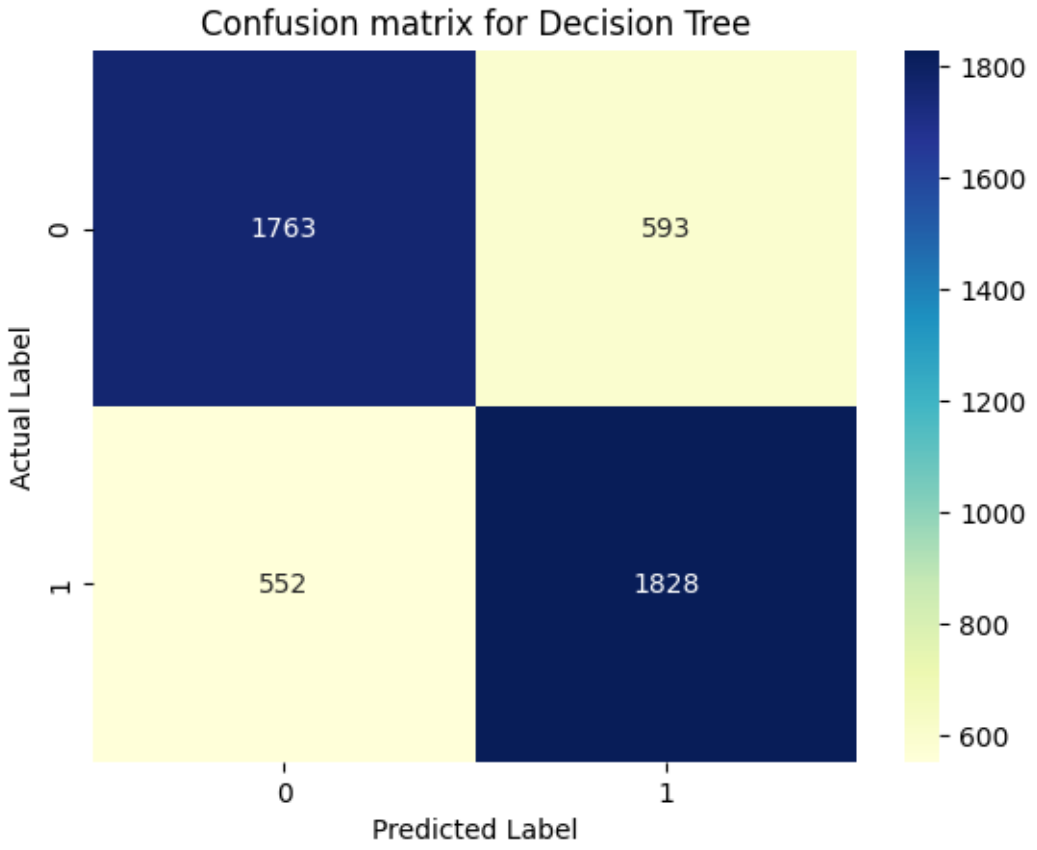
```
Accuracy of Naive Bayes for training database is: 0.6366763215061549
```

[68]:
```
print ("Accuracy of Naive Bayes for testing datatse is:", nb_model.
  ↪score(X_test, y_test))
```

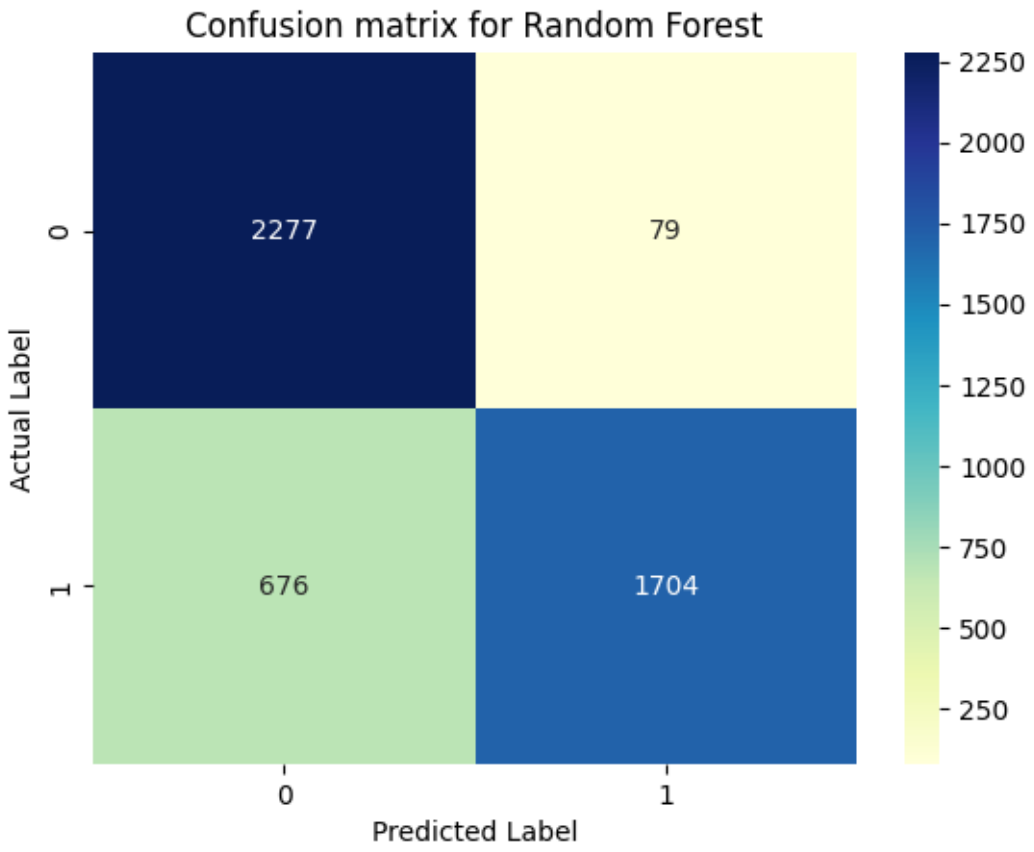
```
Accuracy of Naive Bayes for testing datatse is: 0.620777027027027
```

Naive Bayes thus doesn't overfit. Underfits actually and thus has poor accuracy for both train and test datasets.

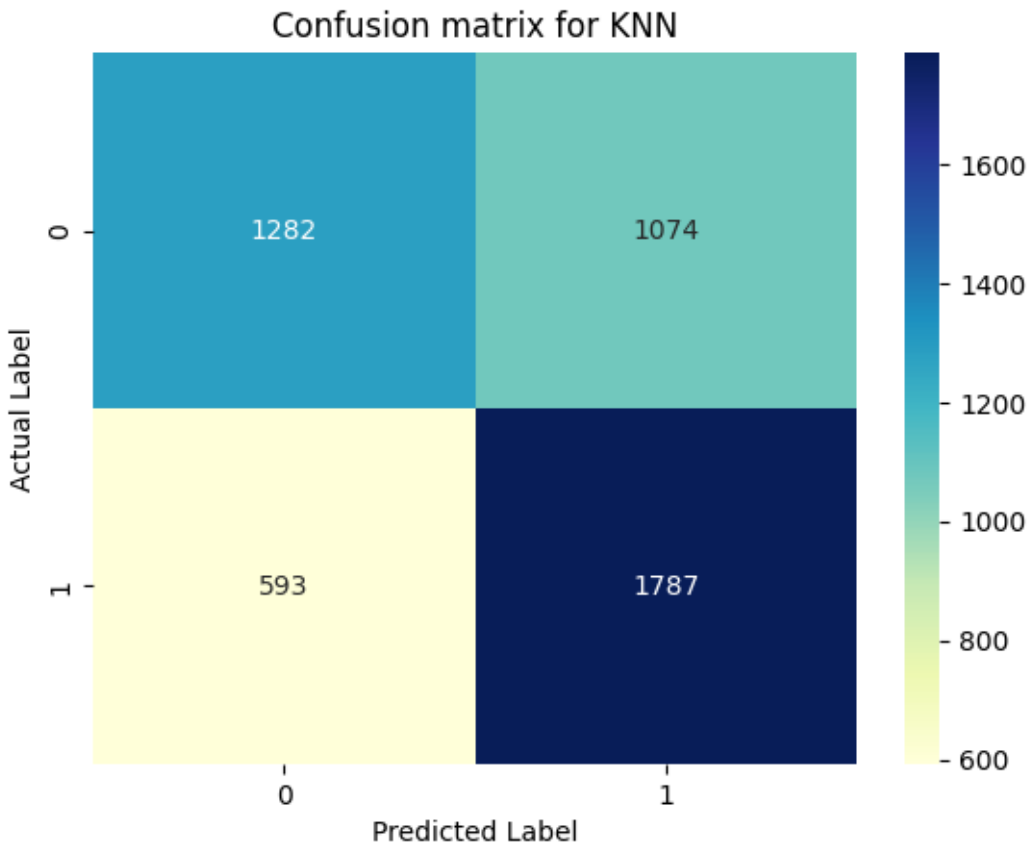
[70]:
```
# confusion matrix for Decision Tree

sns.heatmap(cm2, annot=True, cmap='YlGnBu', fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.title('Confusion matrix for Decision Tree');
```



[71]:
```
# confusion matrix for Random Forest

sns.heatmap(cm3, annot=True, cmap='YlGnBu', fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.title('Confusion matrix for Random Forest');
```



[72]:
```
# confusion matrix for KNN

sns.heatmap(cm4, annot=True, cmap='YlGnBu', fmt='g')
plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
plt.title('Confusion matrix for KNN');
```



[73]:
```
# Random forest has the best performance. let's put all its measures in a report

print(classification_report(y_test, y_pred_rf))
```

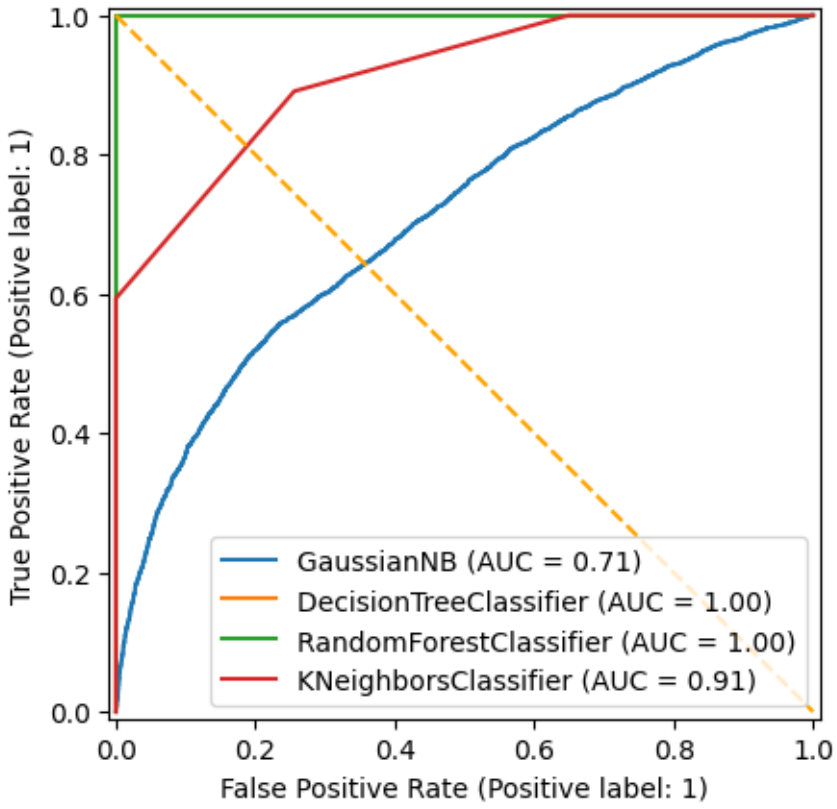
```
              precision    recall  f1-score   support

           0       0.77      0.97      0.86      2356
           1       0.96      0.72      0.82      2380

    accuracy                           0.84      4736
   macro avg       0.86      0.84      0.84      4736
weighted avg       0.86      0.84      0.84      4736
```

[ ]:
```

```

Plot ROC curve for Xtrain and Xtest

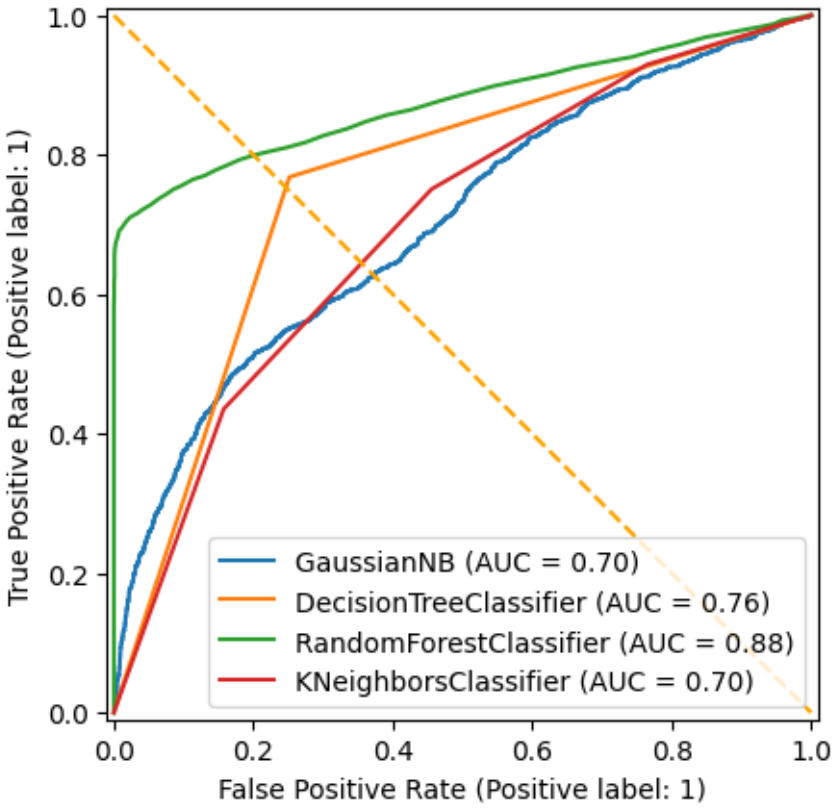
[75]:
```
# ROC Curve for Training dataset

from sklearn.metrics import RocCurveDisplay
```

```
disp =RocCurveDisplay.from_estimator(nb_model, X_train, y_train)
RocCurveDisplay.from_estimator(dt_model, X_train, y_train, ax=disp.ax_)
RocCurveDisplay.from_estimator(rf_model, X_train, y_train, ax=disp.ax_)
RocCurveDisplay.from_estimator(knn_model, X_train, y_train, ax=disp.ax_)
plt.plot([0,1], [1,0], color='orange', linestyle='--');
```



[ ]:

[76]:
```
# ROC Curve for Testing dataset

disp =RocCurveDisplay.from_estimator(nb_model, X_test, y_test)
RocCurveDisplay.from_estimator(dt_model, X_test, y_test, ax=disp.ax_)
RocCurveDisplay.from_estimator(rf_model, X_test, y_test, ax=disp.ax_)
RocCurveDisplay.from_estimator(knn_model, X_test, y_test, ax=disp.ax_)
plt.plot([0,1], [1,0], color='orange', linestyle='--');
```



[ ]:

Evaluate model for new dataset

[78]:
```
# Given 3 new customers
customer_df = pd.DataFrame({
    'CustomerID': [10001, 10002, 10003],
    'Age': [43.296, 55.704, 35.928],
    'Subscription_Length_Months': [33.66, 48.45, 27.08],
    'Monthly_Bill': [107.8, 144.6, 71.65],
    'Total_Usage_GB': [230.95, 398.05, 131.65],
    'Support_Calls': [1, 9, 1],
    'Contract_Type': ['Month-to-Month', 'One Year', 'Month-to-Month'],
    'Payment_Method': ['Debit Card', 'Credit Card', 'PayPal'],
    'Has_Additional_Services': [1, 1, 0]
})
```

[79]:
```
customer_df
```

[79]:
```
   CustomerID     Age  Subscription_Length_Months  Monthly_Bill  \
0       10001  43.296                       33.66        107.80
```

```
1       10002  55.704                       48.45        144.60
2       10003  35.928                       27.08         71.65

   Total_Usage_GB  Support_Calls   Contract_Type Payment_Method  \
0          230.95              1  Month-to-Month     Debit Card
1          398.05              9        One Year    Credit Card
2          131.65              1  Month-to-Month         PayPal

   Has_Additional_Services
0                        1
1                        1
2                        0
```

[80]:
```
# Make a new prediction for new customers (customer_df)

# load new observations (new customers data normalized and encoded)
new_data = pd.DataFrame({
    'CustomerID': [10001, 10002, 10003],
    'Age': [-0.142, 0.892, -0.756],
    'Subscription_Length_Months': [-0.234, 1.245, -0.892],
    'Monthly_Bill': [0.156, 0.892, -0.567],
    'Total_Usage_GB': [-0.127, 0.987, -0.789],
    'Support_Calls': [0, 1, 0],
    'Contract_Type': [0, 1, 0],
    'Payment_Method': [1, 0, 2],
    'Has_Additional_Services': [1, 1, 0]
})
```

[81]:
```
new_data.head(5)
```

[81]:
```
   CustomerID    Age  Subscription_Length_Months  Monthly_Bill  \
0       10001 -0.142                      -0.234         0.156
1       10002  0.892                       1.245         0.892
2       10003 -0.756                      -0.892        -0.567

   Total_Usage_GB  Support_Calls  Contract_Type  Payment_Method  \
0          -0.127              0              0               1
1           0.987              1              1               0
2          -0.789              0              0               2

   Has_Additional_Services
0                        1
1                        1
2                        0
```

[82]:
```
# Create new data with exact same structure as X_train
new_data_standardized = pd.DataFrame({
```

```
    'CustomerID': [10001, 10002, 10003],
    'Age': [-0.142, 0.892, -0.756],
    'Subscription_Length_Months': [-0.234, 1.245, -0.892],
    'Monthly_Bill': [0.156, 0.892, -0.567],
    'Total_Usage_GB': [-0.127, 0.987, -0.789],
    'Support_Calls': [0, 1, 0],
    'Contract_Type': [0, 1, 0],
    'Payment_Method': [1, 0, 2],
    'Has_Additional_Services': [1, 1, 0]
})

# Make predictions
y_pred_new = rf_model.predict(new_data_standardized)
print("Predictions for new customers:", y_pred_new)

# Get prediction probabilities (optional)
y_pred_prob = rf_model.predict_proba(new_data_standardized)
print("\nPrediction probabilities:")
for i, prob in enumerate(y_pred_prob):
    print(f"Customer {i+1} - No Churn: {prob[0]:.2f}, Churn: {prob[1]:.2f}")
```

```
Predictions for new customers: [0 0 0]

Prediction probabilities:
Customer 1 - No Churn: 0.56, Churn: 0.44
Customer 2 - No Churn: 0.68, Churn: 0.32
Customer 3 - No Churn: 0.65, Churn: 0.35
```

[83]:
```
y_pred_new_observation=rf_model.predict(new_data)
```

[84]:
```
y_pred_new_observation
```

[84]:
```
array([0, 0, 0])
```

[ ]:
```

```

Find the Importance of Features

[86]:
```
# ----- Features importance using Random Forest

rf=RandomForestClassifier(random_state=1, n_estimators=100)
```

[87]:
```
rf.fit(X,y)
```

[87]:
```
RandomForestClassifier(random_state=1)
```

[88]:
```
I = rf.feature_importances_
```

[89]:
```
I
```

[89]:
```
array([0.18267379, 0.13463516, 0.13316602, 0.18010421, 0.18317085,
       0.07943338, 0.03809709, 0.04785989, 0.02085962])
```

[90]:
```
df3.columns
```

[90]:
```
Index(['CustomerID', 'Age', 'Subscription_Length_Months', 'Monthly_Bill',
       'Total_Usage_GB', 'Support_Calls', 'Contract_Type', 'Payment_Method',
       'Has_Additional_Services', 'Churn'],
      dtype='object')
```

[91]:
```
# we need the name of features columns. df3.columns include Churn name. we need␣
 ↪to drop Churn

# Define a variable that contains only the feature column names
# First drop CustomerID since it adds no value to target prediction
#df_no_id = df3.drop(['CustomerID'], axis=1)

# Then drop Churn from df_no_id
name_features = df3.drop(df3['Churn']).columns.tolist()
name_features
```

[91]:
```
['CustomerID',
 'Age',
 'Subscription_Length_Months',
 'Monthly_Bill',
 'Total_Usage_GB',
 'Support_Calls',
 'Contract_Type',
 'Payment_Method',
 'Has_Additional_Services',
 'Churn']
```

[92]:
```
len(name_features)
```

[92]:
```
10
```

[93]:
```
len(I)
```

[93]:
```
9
```

[94]:
```
# generate a new dataframe call it df_I

df_I=pd.DataFrame([I, name_features])
```

[95]:
```
df_I
```

[95]:

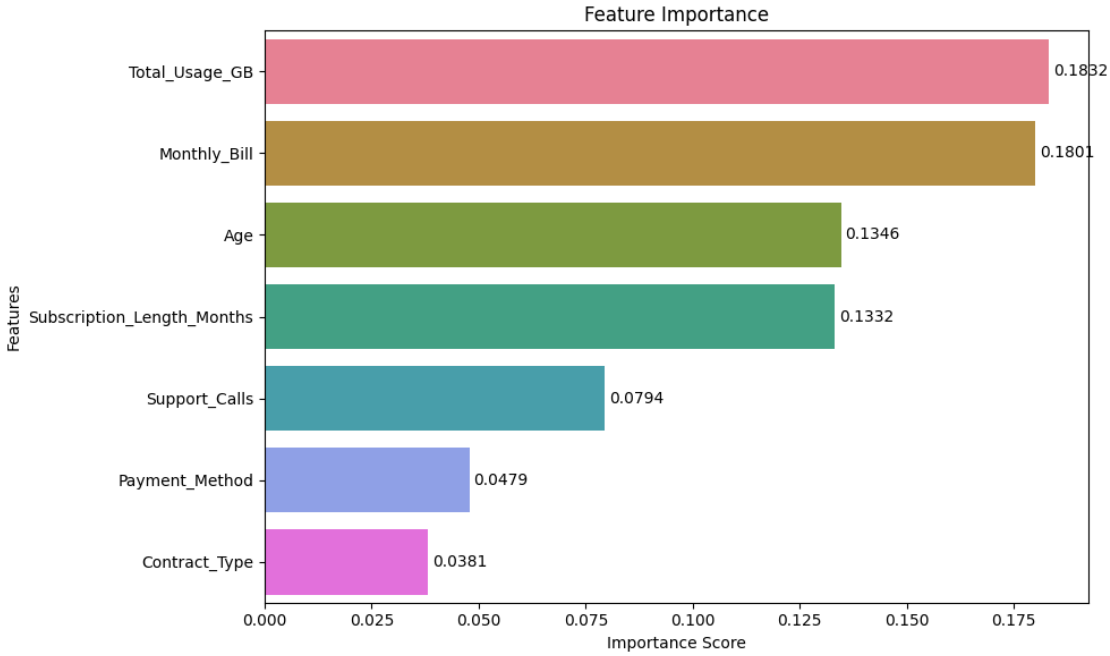
	0	1	2	3	\
0	0.182674	0.134635	0.133166	0.180104	

```
1  CustomerID       Age  Subscription_Length_Months  Monthly_Bill

                4             5             6               7  \
0        0.183171      0.079433      0.038097         0.04786
1  Total_Usage_GB  Support_Calls  Contract_Type  Payment_Method

                        8      9
0                 0.02086   None
1  Has_Additional_Services  Churn
```

[96]:
```python
# Remove 'Churn' and 'CustomerID' from name_features
name_features = name_features[:-1]  # Remove the last element (Churn)
name_features_filtered = [feature for feature in name_features[:-1] if feature !
 ↪= 'CustomerID'] # Remove CustomerID as it isn't a target determining␣
 ↪attribute

# Remove the first element in I (corresponding to CustomerID)
I_filtered = I[1:-1]

# Create the DataFrame
feature_importance_df = pd.DataFrame({
    'Feature': name_features_filtered,
    'Importance': I_filtered
})

# Sort by importance value
feature_importance_df = feature_importance_df.sort_values('Importance',␣
 ↪ascending=False)

# Create the barplot with different colors
plt.figure(figsize=(10, 6))

# Option 1: Using a colorful seaborn palette
colors = sns.color_palette("husl", len(feature_importance_df))  # Create a␣
 ↪colorful palette

# Create the plot with different colors for each bar
ax = sns.barplot(
    data=feature_importance_df,
    x='Importance',
    y='Feature',
    palette=colors
)

# Add value labels to the bars for better readability
for i, v in enumerate(feature_importance_df['Importance']):
    ax.text(v + 0.001, i, f"{v:.4f}", va='center')
```

```
plt.title('Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.tight_layout()
plt.show()
```

```
/var/folders/9p/02j0tmhs5w35zmgt4_pc9src0000gn/T/ipykernel_4362/2523790702.py:24
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  ax = sns.barplot(
```



[ ]:

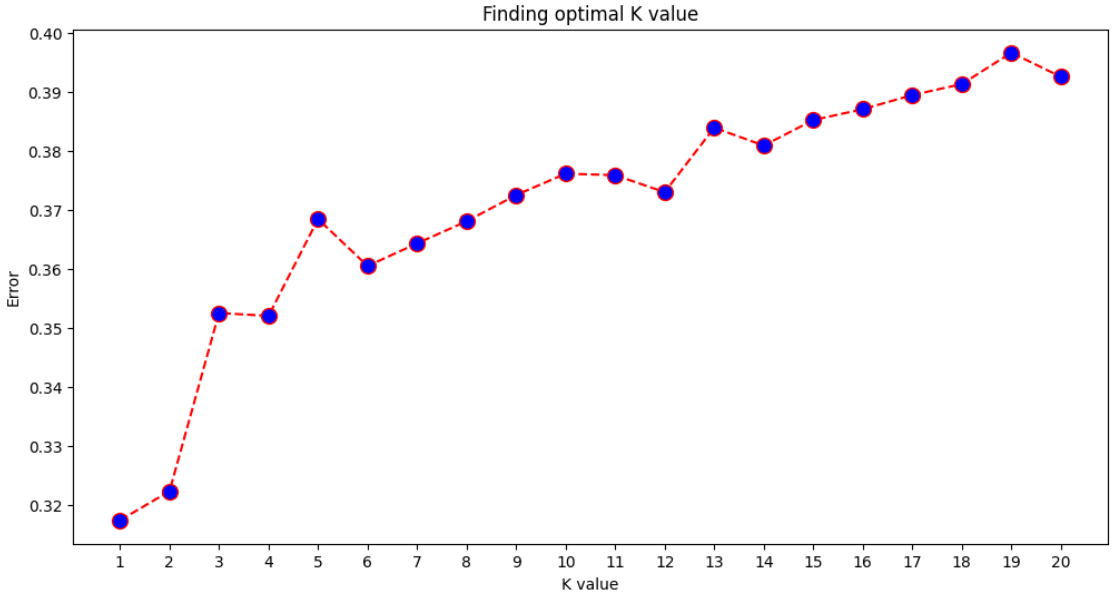
Find the best value of K in KNN

[98]:
```
from sklearn.metrics import balanced_accuracy_score
```

[99]:
```
error=[]

#calculating the error for k values between 1 and 39
```

```
for i in range(1,21):  # i=1,2,...,20
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    accuracy = balanced_accuracy_score(y_test, pred_i)
    error.append(1-accuracy)

error;
```

[100]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))

plt.plot(range(1,21), error, color='red',   linestyle='dashed', marker='o',␣
 ↪markerfacecolor='blue', markersize=10)

plt.title ('Finding optimal K value')
plt.xlabel ('K value')
plt.ylabel('Error');

# Set the ticks on the X-axis to integer values
plt.xticks(ticks=range(1, 21));  # Set the ticks to integers 1 to 20
```



Remodeling KNN using the optimal K value

[102]:
```
# KNN model with n=1
```

```
knn_model =KNeighborsClassifier(n_neighbors=1)
# For n_neighbors =3, accuracy is 0.74
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
evaluate_model(y_test, y_pred_knn, "KNN")
```

```
Model: KNN
Confusion Matrix:
[[1313 1043]
 [ 457 1923]]
Accuracy:  0.68
Precision: 0.65
PRecall:   0.81
F1 Score:  0.72
```

[103]:
```
# KNN model with n=2

knn_model =KNeighborsClassifier(n_neighbors=2)
# For n_neighbors =3, accuracy is 0.74
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
evaluate_model(y_test, y_pred_knn, "KNN")
```

```
Model: KNN
Confusion Matrix:
[[1758  598]
 [ 930 1450]]
Accuracy:  0.68
Precision: 0.71
PRecall:   0.61
F1 Score:  0.65
```

[104]:
```
# KNN model with n=4

knn_model =KNeighborsClassifier(n_neighbors=4)
# For n_neighbors =3, accuracy is 0.74
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
evaluate_model(y_test, y_pred_knn, "KNN")
```

```
Model: KNN
Confusion Matrix:
[[1611  745]
 [ 923 1457]]
Accuracy:  0.65
Precision: 0.66
PRecall:   0.61
```

```
F1 Score:  0.64
```

As **n** increases in KNN, the model's performance varies. At **n=1**, the model achieves the highest recall (0.95) but at the cost of lower precision (0.74), indicating a tendency to favor positive predictions. While accuracy remains relatively stable (~0.74–0.81) across different values of **n**, a trade-off between precision and recall becomes evident. **n=2** offers the best balance (Precision: 0.78, Recall: 0.83, F1: 0.80). However, performance declines for **n=3** and **n=4**, suggesting that increasing the number of neighbors beyond **n=2** diminishes the model's effectiveness.

[ ]:

[ ]:

Conditional Probabilities

[107]:
```
# 1. What is the probability of churn for customers with a Two Year contract␣
 ↪and subscription length over 36 months?
# P(Churn = 1 | Contract_Type = "Two Year", Subscription_Length_Months > 36)
((df['Contract_Type'] == "Two Year") & (df['Subscription_Length_Months'] > 36)␣
 ↪& (df['Churn'] == 1)).sum() / ((df['Contract_Type'] == "Two Year") &␣
 ↪(df['Subscription_Length_Months'] > 36)).sum()
```

[107]:
```
0.2153846153846154
```

[ ]:

[108]:
```
# 2. What is the probability of churn for customers with usage over 300GB and␣
 ↪monthly bill over $150?
# P(Churn = 1 | Total_Usage_GB > 300, Monthly_Bill > 150)
((df['Total_Usage_GB'] > 300) & (df['Monthly_Bill'] > 150) & (df['Churn'] ==␣
 ↪1)).sum() / ((df['Total_Usage_GB'] > 300) & (df['Monthly_Bill'] > 150)).sum()
```

[108]:
```
0.20408163265306123
```

[ ]:

[109]:
```
# 3. What is the probability of churn for customers with more than 5 support␣
 ↪calls?
# P(Churn = 1 | Support_Calls > 5)
((df['Support_Calls'] > 5) & (df['Churn'] == 1)).sum() / (df['Support_Calls'] >␣
 ↪5).sum()
```

[109]:
```
0.20586062546170894
```

[ ]:

[110]:
```
# 4. What is the probability of churn for customers aged 60+ with low usage␣
 ↪(<100GB)?
```

```
# P(Churn = 1 | Age >= 60, Total_Usage_GB < 100)
((df['Age'] >= 60) & (df['Total_Usage_GB'] < 100) & (df['Churn'] == 1)).sum() /␣
↪((df['Age'] >= 60) & (df['Total_Usage_GB'] < 100)).sum()
```

[110]:
```
0.18495934959349594
```

[ ]:
```

```

[111]:
```
# 5. What is the probability of churn for customers with monthly bill over $90?
# P(Churn = 1 | Monthly_Bill > 90)
((df['Monthly_Bill'] > 90) & (df['Churn'] == 1)).sum() / (df['Monthly_Bill'] >␣
↪90).sum()
```

[111]:
```
0.20373277516134658
```

[ ]:
```

```

[112]:
```
# 6. What is the probability that a heavy usage customer (>300GB) has a␣
↪Month-to-Month contract?
# P(Contract_Type = "Month-to-Month" | Total_Usage_GB > 300)
((df['Total_Usage_GB'] > 300) & (df['Contract_Type'] == "Month-to-Month")).
↪sum() / (df['Total_Usage_GB'] > 300).sum()
```

[112]:
```
0.33398821218074654
```

[ ]:
```

```

[113]:
```
# 7. What is the probability of churn for customers with more than 5 support␣
↪calls, a Month-to-Month contract, and monthly bill over $100?
# P(Churn = 1 | Support_Calls > 5, Contract_Type = "Month-to-Month",␣
↪Monthly_Bill > 100)
((df['Support_Calls'] > 5) & (df['Contract_Type'] == "Month-to-Month") &␣
↪(df['Monthly_Bill'] > 100) & (df['Churn'] == 1)).sum() /␣
↪((df['Support_Calls'] > 5) & (df['Contract_Type'] == "Month-to-Month") &␣
↪(df['Monthly_Bill'] > 100)).sum()
```

[113]:
```
0.20084865629420084
```

[ ]:
```

```

Feature Importance using Naive Bayes

[115]:
```
# Determining the significance of Monthly Bill when predicting customer churn
# (Churn = 1) as the target variable.

# Conditional probabilities for high vs low monthly bill
median_bill = df['Monthly_Bill'].median()
```

```
prob_high_bill = ((df['Monthly_Bill'] > median_bill) & (df['Churn'] == 1)).
 ↪sum() / (df['Monthly_Bill'] > median_bill).sum()
prob_low_bill = ((df['Monthly_Bill'] <= median_bill) & (df['Churn'] == 1)).
 ↪sum() / (df['Monthly_Bill'] <= median_bill).sum()

print("Conditional probabilities:", "\nHigh_Bill:", prob_high_bill, "\nLow_Bill:
 ↪", prob_low_bill)

# Calculating difference to determine significance
bill_difference = abs(prob_high_bill - prob_low_bill)
print("Feature importance of Monthly Bill is:")
print("Difference:", bill_difference)
```

```
Conditional probabilities:
High_Bill: 0.2002
Low_Bill: 0.2212
Feature importance of Monthly Bill is:
Difference: 0.02100000000000002
```

[ ]:

[116]:
```
# Determining the significance of Total Usage when predicting customer churn
median_usage = df['Total_Usage_GB'].median()
prob_high_usage = ((df['Total_Usage_GB'] > median_usage) & (df['Churn'] == 1)).
 ↪sum() / (df['Total_Usage_GB'] > median_usage).sum()
prob_low_usage = ((df['Total_Usage_GB'] <= median_usage) & (df['Churn'] == 1)).
 ↪sum() / (df['Total_Usage_GB'] <= median_usage).sum()

print("\nConditional probabilities:", "High_Usage:", prob_high_usage,␣
 ↪"Low_Usage:", prob_low_usage)

# Calculating difference to determine significance
usage_difference = abs(prob_high_usage - prob_low_usage)
print("Feature importance of Total Usage is:")
print("Difference:", usage_difference)
```

```
Conditional probabilities: High_Usage: 0.21 Low_Usage: 0.2114
Feature importance of Total Usage is:
Difference: 0.0014000000000000123
```

[ ]:

[117]:
```
# Determining the significance of Age > 30 when predicting customer churn
prob_age_over_30 = ((df['Age'] > 30) & (df['Churn'] == 1)).sum() / (df['Age'] >␣
 ↪30).sum()
prob_age_under_30 = ((df['Age'] <= 30) & (df['Churn'] == 1)).sum() / (df['Age']␣
 ↪<= 30).sum()
```

```
print("\nConditional probabilities:", "Age>30:", prob_age_over_30, "Age<=30:",␣
↪prob_age_under_30)

# Calculating difference to determine significance
age_difference = abs(prob_age_over_30 - prob_age_under_30)
print("Feature importance of Age > 30 is:")
print("Difference:", age_difference)
```

```
Conditional probabilities: Age>30: 0.21192648759799512 Age<=30:
0.206399278954484
Feature importance of Age > 30 is:
Difference: 0.005527208643511111
```

[ ]:

[118]:
```
# Determining the significance of Subscription Length when predicting customer␣
↪churn
median_sub_length = df['Subscription_Length_Months'].median()
prob_long_sub = ((df['Subscription_Length_Months'] > median_sub_length) &␣
↪(df['Churn'] == 1)).sum() / (df['Subscription_Length_Months'] >␣
↪median_sub_length).sum()
prob_short_sub = ((df['Subscription_Length_Months'] <= median_sub_length) &␣
↪(df['Churn'] == 1)).sum() / (df['Subscription_Length_Months'] <=␣
↪median_sub_length).sum()

print("\nConditional probabilities:", "Long_Subscription:", prob_long_sub,␣
↪"Short_Subscription:", prob_short_sub)

# Calculating difference to determine significance
sub_difference = abs(prob_long_sub - prob_short_sub)
print("Feature importance of Subscription Length is:")
print("Difference:", sub_difference)
```

```
Conditional probabilities: Long_Subscription: 0.2132132132132132
Short_Subscription: 0.2081918081918082
Feature importance of Subscription Length is:
Difference: 0.005021405021405023
```

[ ]:

[119]:
```
# Comparing feature importances
features = ['Monthly_Bill', 'Total_Usage_GB', 'Age > 30', 'Subscription_Length']
importances = [bill_difference, usage_difference, age_difference,␣
↪sub_difference]
```

```
# Print ranked list of features by importance
for feature, importance in sorted(zip(features, importances), key=lambda x:␣
 ↪x[1], reverse=True):
    print(f"{feature}: {importance:.4f}")
```

```
Monthly_Bill: 0.0210
Age > 30: 0.0055
Subscription_Length: 0.0050
Total_Usage_GB: 0.0014
```

[ ]:

RegressionAnalysis

March 3, 2025

1 Regression Analysis

[2]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import r2_score, mean_squared_error
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from statsmodels.stats.outliers_influence import variance_inflation_factor
from lazypredict.Supervised import LazyRegressor
```

DATASET: House Rent Dataset FROM Kaggle.com

[3]:
```
# ------------------------------------
# Load and preprocess dataset
# ------------------------------------
df = pd.read_csv(r'House_Rent_Dataset.csv')
```

Data Preprocessing

[4]:
```
# ------------------------------
#  Checking for missing values
# ------------------------------
missing_values = df.isnull().sum()
print("Missing values:\n", missing_values)
```

```
Missing values:
 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
```

[5]:
```python
def convert_to_floor_number(floor_str):
    """
    Convert floor string to numeric value
    -2: Lower Basement
    -1: Upper Basement
    0: Ground Floor
    1+: Regular floors
    """
    if isinstance(floor_str, str):
        if 'Upper Basement' in floor_str:
            return -1
        elif 'Lower Basement' in floor_str:
            return -2
        elif 'Ground' in floor_str:
            return 0
        else:
            try:
                return int(floor_str.split()[0])
            except:
                return None
    return None

# Add floor_number feature
df['floor_number'] = df['Floor'].apply(convert_to_floor_number)

# Display first few rows to verify
print(df['floor_number'].head())

# Show value counts to verify distribution
print("\nFloor number distribution:")
print(df['floor_number'].value_counts().sort_index())
```

```
0    0
1    1
2    1
3    1
4    1
Name: floor_number, dtype: int64
```

Floor number distribution:

floor_number	
-2	11
-1	23
0	927
1	1161
2	945
3	512
4	272
5	164
6	93
7	74
8	66
9	65
10	67
11	43
12	47
13	15
14	34
15	41
16	21
17	22
18	26
19	16
20	12
21	6
22	3
23	9
24	6
25	12
26	3
27	3
28	4
29	1
30	5
32	3
33	1
34	4
35	3
36	2
37	1
39	1
40	2
41	1
43	1
44	1
45	2
46	1

```
47       1
48       2
49       1
50       1
53       2
60       3
65       3
76       1
Name: count, dtype: int64
```

[7]:
```
# Drop unnecessary columns
df_cleaned = df.drop(columns=['Posted On', 'Floor', 'Area Locality',
                              'Tenant Preferred', 'Point of Contact'])
```

[12]:
```
# Convert categorical variables into numeric format
furnishing_mapping = {'Unfurnished': 0, 'Semi-Furnished': 1, 'Furnished': 2}
df_cleaned['Furnishing Status'] = df_cleaned['Furnishing Status'].
 ↪map(furnishing_mapping)

df_cleaned = pd.get_dummies(df_cleaned, columns=['Area Type', 'City'],␣
 ↪drop_first=True)

# Ensure all columns are numeric
df_cleaned = df_cleaned.apply(pd.to_numeric, errors='coerce')

df_cleaned.head()
```

[12]:

	BHK	Rent	Size	Furnishing Status	Bathroom	floor_number	\
0	2	10000	1100	0	2	0	
1	2	20000	800	1	1	1	
2	2	17000	1000	1	1	1	
3	2	10000	800	0	1	1	
4	2	7500	850	0	1	1	

	Area Type_Carpet Area	Area Type_Super Area	City_Chennai	City_Delhi	\
0	False	True	False	False	
1	False	True	False	False	
2	False	True	False	False	
3	False	True	False	False	
4	True	False	False	False	

	City_Hyderabad	City_Kolkata	City_Mumbai
0	False	True	False
1	False	True	False
2	False	True	False
3	False	True	False
4	False	True	False

[14]:
```
# Convert boolean columns to integers (for OLS)
for col in df_cleaned.columns:
    if df_cleaned[col].dtype == 'bool':
        df_cleaned[col] = df_cleaned[col].astype(int)
```

[16]:
```
# -------------------------------
# Define dependent and independent variables
# -------------------------------
X = df_cleaned.drop('Rent', axis=1)
y = df_cleaned['Rent']
```

Generating Models

[18]:
```
# -------------------------------
#  Split the data into training and testing sets
# -------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

[20]:
```
# -------------------------------
#  Multiple Linear Regression using sklearn
# -------------------------------
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

print("=== Multiple Linear Regression ===")
r2=r2_score(y_test, y_pred)
print("R2:", r2)
print("MSE:", mean_squared_error(y_test, y_pred))
# Calculating Adjusted R²
n = X_test.shape[0]  # Number of samples
p = X_test.shape[1]  # Number of predictors
adjusted_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1))
print("Adjusted R2:",adjusted_r2)

# -------------------------------
#  Polynomial Regression (e.g., degree=2)
# -------------------------------
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)

lr_poly = LinearRegression()
lr_poly.fit(X_poly_train, y_train)
y_poly_pred = lr_poly.predict(X_poly_test)

r2=r2_score(y_test, y_poly_pred)
```

```
print("=== Polynomial Regression (degree=2) ===")
print("R2:", r2)
print("MSE:", mean_squared_error(y_test, y_poly_pred))

# Calculating Adjusted R²
n = X_poly_test.shape[0]  # Number of samples
p = X_poly_test.shape[1]  # Number of predictors
adjusted_r2 = 1 - ((1 - r2) * (n - 1) / (n - p - 1))
print("Adjusted R2:",adjusted_r2)

print("Interpretation: Since the R2 score is higher for polynomial regression␣
 ↪than for multiple linear regression, the model follows a polynomial curve.")
# Extract coefficients and intercept for polynomial regression
coefficients_poly = lr_poly.coef_
intercept_poly = lr_poly.intercept_

# Get feature names
feature_names = poly.get_feature_names_out(X_train.columns)
#print(feature_names)

# Construct the equation
equation = f"Rent = {intercept_poly:.2f}"
for i, coef in enumerate(coefficients_poly):
    equation += f" + {coef:.2f} * {feature_names[i]}"

print("=== Equation of the Model ===")
print(equation)

# ------------------------------
#  LazyPredict: Evaluate multiple regression models
# ------------------------------
reg = LazyRegressor(verbose=0, ignore_warnings=True, custom_metric=None)
models, predictions = reg.fit(X_train, X_test, y_train, y_test)
print("=== LazyPredict Model Comparison ===")
print(models)
```

```
=== Multiple Linear Regression ===
R2: 0.5228843329265807
MSE: 1901487865.6850293
Adjusted R2: 0.5167739935403682
=== Polynomial Regression (degree=2) ===
R2: 0.8137733813128744
MSE: 742184086.8760061
Adjusted R2: 0.7942618613107308
Interpretation: Since the R2 score is higher for polynomial regression than for
multiple linear regression, the model follows a polynomial curve.
=== Equation of the Model ===
Rent = 236693.67 + -40788374731.34 * BHK + 81575716.39 * Size + 174315.13 *
```

```
Furnishing Status + 157807.45 * Bathroom + 26404.82 * floor_number + -112617.09
* Area Type_Carpet Area + -114163.93 * Area Type_Super Area + 52728.54 *
City_Chennai + -4197.02 * City_Delhi + -4768.28 * City_Hyderabad + 509.40 *
City_Kolkata + -8347.23 * City_Mumbai + -382.45 * BHK^2 + -4.24 * BHK Size +
-1763.72 * BHK Furnishing Status + 6712.43 * BHK Bathroom + 891.13 * BHK
floor_number + 40788361229.53 * BHK Area Type_Carpet Area + 40788362939.56 * BHK
Area Type_Super Area + 8132.09 * BHK City_Chennai + 3667.74 * BHK City_Delhi +
7881.57 * BHK City_Hyderabad + 10132.94 * BHK City_Kolkata + -20261.69 * BHK
City_Mumbai + 0.00 * Size^2 + 7.13 * Size Furnishing Status + 4.03 * Size
Bathroom + -1.07 * Size floor_number + -81575669.32 * Size Area Type_Carpet Area
+ -81575685.43 * Size Area Type_Super Area + -33.46 * Size City_Chennai + -34.51
* Size City_Delhi + -38.79 * Size City_Hyderabad + -42.21 * Size City_Kolkata +
109.30 * Size City_Mumbai + -682.73 * Furnishing Status^2 + -736.49 * Furnishing
Status Bathroom + 648.78 * Furnishing Status floor_number + -175857.84 *
Furnishing Status Area Type_Carpet Area + -175173.56 * Furnishing Status Area
Type_Super Area + 1054.59 * Furnishing Status City_Chennai + 1934.58 *
Furnishing Status City_Delhi + -679.30 * Furnishing Status City_Hyderabad +
1298.05 * Furnishing Status City_Kolkata + 6327.87 * Furnishing Status
City_Mumbai + -3317.53 * Bathroom^2 + 141.87 * Bathroom floor_number +
-165789.41 * Bathroom Area Type_Carpet Area + -165857.30 * Bathroom Area
Type_Super Area + 4721.69 * Bathroom City_Chennai + 22330.81 * Bathroom
City_Delhi + 1478.58 * Bathroom City_Hyderabad + 5507.58 * Bathroom City_Kolkata
+ 4141.27 * Bathroom City_Mumbai + -14.38 * floor_number^2 + -28583.44 *
floor_number Area Type_Carpet Area + -28223.98 * floor_number Area Type_Super
Area + 768.83 * floor_number City_Chennai + -360.44 * floor_number City_Delhi +
1869.63 * floor_number City_Hyderabad + -113.98 * floor_number City_Kolkata +
1164.90 * floor_number City_Mumbai + -112617.06 * Area Type_Carpet Area^2 + 0.00
* Area Type_Carpet Area Area Type_Super Area + -109708.23 * Area Type_Carpet
Area City_Chennai + -8605.85 * Area Type_Carpet Area City_Delhi + 7524.22 * Area
Type_Carpet Area City_Hyderabad + -4002.08 * Area Type_Carpet Area City_Kolkata
+ 1563.71 * Area Type_Carpet Area City_Mumbai + -114163.93 * Area Type_Super
Area^2 + -100236.42 * Area Type_Super Area City_Chennai + 4408.78 * Area
Type_Super Area City_Delhi + 23599.69 * Area Type_Super Area City_Hyderabad +
4511.47 * Area Type_Super Area City_Kolkata + -9910.94 * Area Type_Super Area
City_Mumbai + 52728.53 * City_Chennai^2 + 0.00 * City_Chennai City_Delhi + 0.00
* City_Chennai City_Hyderabad + 0.00 * City_Chennai City_Kolkata + 0.00 *
City_Chennai City_Mumbai + -4197.07 * City_Delhi^2 + 0.00 * City_Delhi
City_Hyderabad + 0.00 * City_Delhi City_Kolkata + 0.00 * City_Delhi City_Mumbai
+ -4768.27 * City_Hyderabad^2 + 0.00 * City_Hyderabad City_Kolkata + 0.00 *
City_Hyderabad City_Mumbai + 509.39 * City_Kolkata^2 + 0.00 * City_Kolkata
City_Mumbai + -8347.24 * City_Mumbai^2

100%|                        | 42/42 [00:05<00:00,  7.81it/s]

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000177 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 329
```

```
[LightGBM] [Info] Number of data points in the train set: 3796, number of used
features: 12
[LightGBM] [Info] Start training from score 35151.516333
=== LazyPredict Model Comparison ===
```

Model	Adjusted R-Squared	R-Squared	RMSE
PoissonRegressor	0.79	0.79	28664.06
GradientBoostingRegressor	0.68	0.69	35210.52
RandomForestRegressor	0.67	0.68	35880.73
HistGradientBoostingRegressor	0.66	0.67	36347.19
LGBMRegressor	0.64	0.64	37843.00
BaggingRegressor	0.62	0.63	38616.00
AdaBoostRegressor	0.62	0.63	38634.73
XGBRegressor	0.55	0.56	42090.64
GammaRegressor	0.54	0.55	42381.71
LassoLarsIC	0.52	0.52	43604.87
LassoLarsCV	0.52	0.52	43605.25
LarsCV	0.52	0.52	43605.28
LassoCV	0.52	0.52	43605.80
Lasso	0.52	0.52	43606.01
LassoLars	0.52	0.52	43606.02
TransformedTargetRegressor	0.52	0.52	43606.05
LinearRegression	0.52	0.52	43606.05
Ridge	0.52	0.52	43607.47
Lars	0.52	0.52	43610.93
RidgeCV	0.52	0.52	43620.32
BayesianRidge	0.52	0.52	43674.06
SGDRegressor	0.51	0.52	43697.38
OrthogonalMatchingPursuitCV	0.51	0.52	43943.77
ElasticNet	0.47	0.48	45517.64
KNeighborsRegressor	0.45	0.46	46593.13
TweedieRegressor	0.44	0.45	46890.23
DecisionTreeRegressor	0.40	0.40	48719.24
ExtraTreesRegressor	0.38	0.39	49243.16
HuberRegressor	0.36	0.37	50114.02
PassiveAggressiveRegressor	0.34	0.35	50970.66
ExtraTreeRegressor	0.32	0.33	51665.41
OrthogonalMatchingPursuit	0.30	0.31	52344.68
KernelRidge	0.22	0.23	55289.87
ElasticNetCV	0.06	0.07	60886.67
RANSACRegressor	0.05	0.06	61287.57
DummyRegressor	-0.01	-0.00	63134.82
MLPRegressor	-0.03	-0.02	63755.27
NuSVR	-0.05	-0.04	64310.21
SVR	-0.10	-0.08	65658.21
QuantileRegressor	-0.10	-0.08	65746.02
LinearSVR	-0.25	-0.23	70051.32
GaussianProcessRegressor	-43292.31	-42744.87	13052148.71

Model	Time Taken
PoissonRegressor	0.01
GradientBoostingRegressor	0.15
RandomForestRegressor	0.43
HistGradientBoostingRegressor	0.26
LGBMRegressor	0.04
BaggingRegressor	0.04
AdaBoostRegressor	0.07
XGBRegressor	0.27
GammaRegressor	0.02
LassoLarsIC	0.01
LassoLarsCV	0.02
LarsCV	0.01
LassoCV	0.03
Lasso	0.05
LassoLars	0.01
TransformedTargetRegressor	0.00
LinearRegression	0.01
Ridge	0.00
Lars	0.01
RidgeCV	0.00
BayesianRidge	0.01
SGDRegressor	0.00
OrthogonalMatchingPursuitCV	0.01
ElasticNet	0.00
KNeighborsRegressor	0.01
TweedieRegressor	0.00
DecisionTreeRegressor	0.01
ExtraTreesRegressor	0.36
HuberRegressor	0.05
PassiveAggressiveRegressor	0.01
ExtraTreeRegressor	0.01
OrthogonalMatchingPursuit	0.00
KernelRidge	0.33
ElasticNetCV	0.04
RANSACRegressor	0.05
DummyRegressor	0.00
MLPRegressor	1.22
NuSVR	0.26
SVR	0.35
QuantileRegressor	0.31
LinearSVR	0.00
GaussianProcessRegressor	0.88

[21]:
```
# ----------------------------------
# Ensure all data is numeric
# ----------------------------------
print("Final Data Types of X_train:\n", X_train.dtypes)
```

```
Final Data Types of X_train:
 BHK                        int64
Size                       int64
Furnishing Status          int64
Bathroom                   int64
floor_number               int64
Area Type_Carpet Area      int64
Area Type_Super Area       int64
City_Chennai               int64
City_Delhi                 int64
City_Hyderabad             int64
City_Kolkata               int64
City_Mumbai                int64
dtype: object
```

[22]:
```
# ------------------------------
#  Checking assumptions using statsmodels (for Multiple Linear Regression)
# ------------------------------
X_train_sm = sm.add_constant(X_train)  # Add constant for intercept
model_sm = sm.OLS(y_train, X_train_sm).fit()

# Fit the model using statsmodels to get a detailed summary (including p-values)
print("=== Statsmodels Regression Summary ===")
print(model_sm.summary())

# P-values for the coefficients
print("P-values for coefficients:")
print(model_sm.pvalues)
print()
```

```
=== Statsmodels Regression Summary ===
                            OLS Regression Results
==============================================================================
Dep. Variable:                   Rent   R-squared:                       0.288
Model:                            OLS   Adj. R-squared:                  0.286
Method:                 Least Squares   F-statistic:                     127.7
Date:                Sun, 02 Mar 2025   Prob (F-statistic):          2.07e-268
Time:                        23:57:11   Log-Likelihood:                -47664.
No. Observations:                3796   AIC:                         9.535e+04
Df Residuals:                    3783   BIC:                         9.543e+04
Df Model:                          12
Covariance Type:            nonrobust
==============================================================================
```

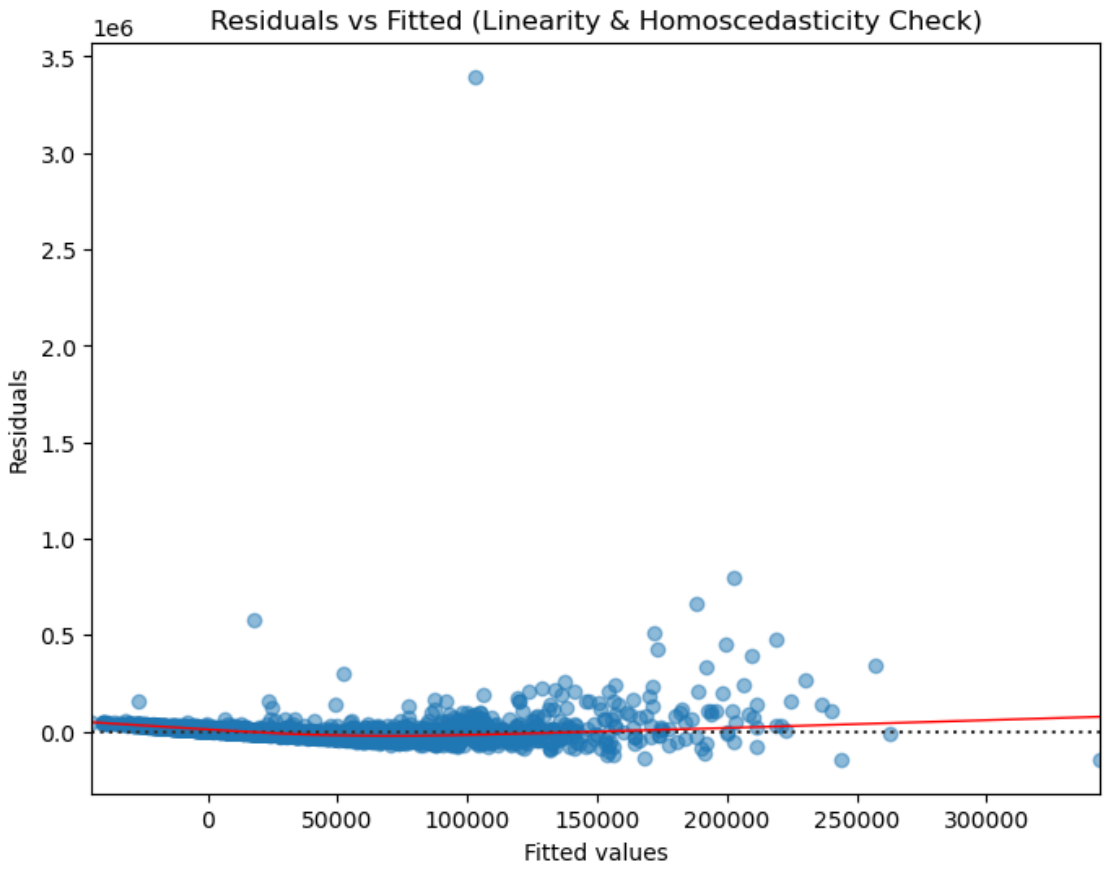
```
=========
                             coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
---------
const                  -3.186e+04   4.88e+04     -0.653      0.514   -1.28e+05
6.38e+04
BHK                     2859.5472   2387.188      1.198      0.231   -1820.752
7539.847
Size                      37.2556      2.920     12.758      0.000      31.530
42.981
Furnishing Status       2829.7975   1686.370      1.678      0.093    -476.485
6136.080
Bathroom                1.021e+04   2510.059      4.069      0.000    5292.992
1.51e+04
floor_number            1369.5544    243.030      5.635      0.000     893.071
1846.038
Area Type_Carpet Area -5617.1468   4.87e+04     -0.115      0.908   -1.01e+05
8.99e+04
Area Type_Super Area  -1.058e+04   4.87e+04     -0.217      0.828   -1.06e+05
8.49e+04
City_Chennai          -7493.7050   3660.268     -2.047      0.041   -1.47e+04
-317.416
City_Delhi              1.093e+04   4133.658      2.643      0.008    2821.728
1.9e+04
City_Hyderabad        -1.676e+04   3725.922     -4.498      0.000   -2.41e+04
-9454.794
City_Kolkata           -353.8969   4411.010     -0.080      0.936   -9002.084
8294.290
City_Mumbai             4.607e+04   4147.924     11.108      0.000    3.79e+04
5.42e+04
==============================================================================
Omnibus:                    10361.874   Durbin-Watson:                   2.013
Prob(Omnibus):                  0.000   Jarque-Bera (JB):        397954117.987
Skew:                          33.316   Prob(JB):                         0.00
Kurtosis:                    1587.804   Cond. No.                     8.74e+04
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[2] The condition number is large, 8.74e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
P-values for coefficients:
const                  0.51
BHK                    0.23
Size                   0.00
Furnishing Status      0.09
```

```
Bathroom                0.00
floor_number            0.00
Area Type_Carpet Area   0.91
Area Type_Super Area    0.83
City_Chennai            0.04
City_Delhi              0.01
City_Hyderabad          0.00
City_Kolkata            0.94
City_Mumbai             0.00
dtype: float64
```

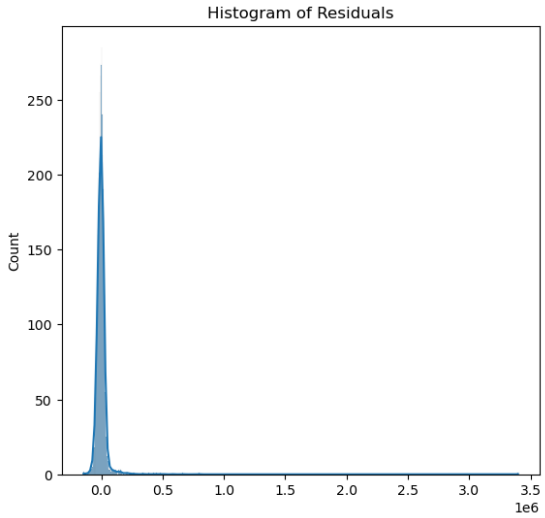
Performance Analysis

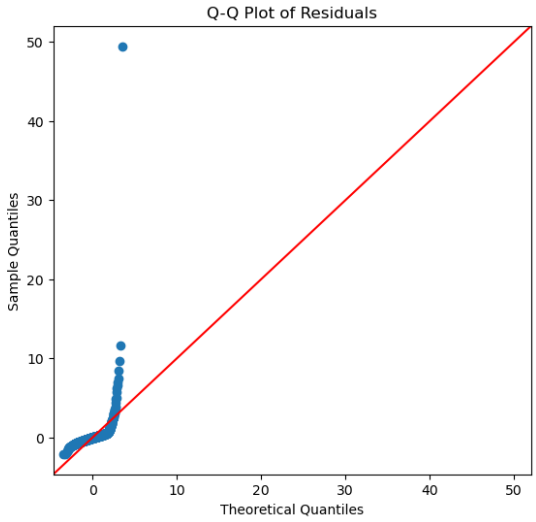
[23]:
```
# ------------------------------
# Diagnostic Plots and Assumptions Check
# ------------------------------
# Get residuals and fitted values from the statsmodels fit
residuals = model_sm.resid
fitted = model_sm.fittedvalues
```

[24]:
```
# Linearity & Constant Variance (Homoscedasticity):
plt.figure(figsize=(8, 6))
sns.residplot(x=fitted, y=y_train, lowess=True,
 line_kws={'color': 'red', 'lw': 1}, scatter_kws={'alpha':0.5})
plt.xlabel("Fitted values")
plt.ylabel("Residuals")
plt.title("Residuals vs Fitted (Linearity & Homoscedasticity Check)")
plt.show()
```



[43]:
```
# Normality of Errors: Histogram and Q-Q plot
plt.figure(figsize=(14, 6))
plt.subplot(1, 2, 1)
sns.histplot(residuals, kde=True)
plt.title("Histogram of Residuals")
ax=plt.subplot(1, 2, 2)
sm.qqplot(residuals, line='45', fit=True, ax=ax)
plt.title("Q-Q Plot of Residuals")
plt.show()
```





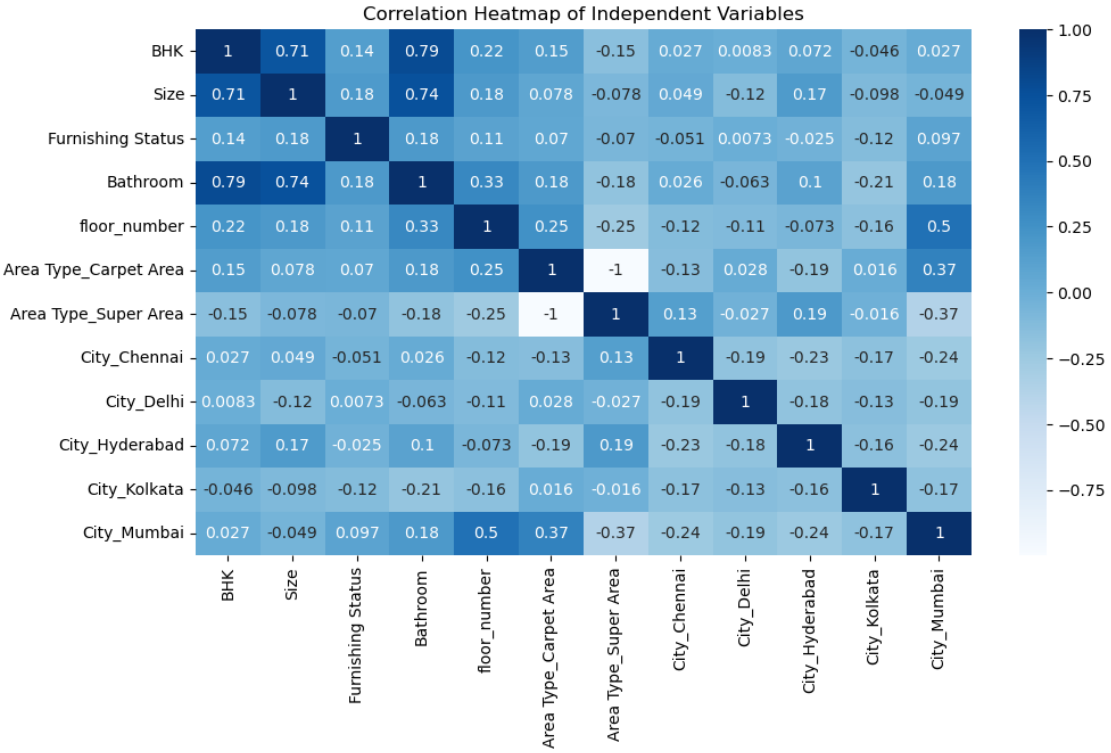
Feature Dependence

[26]:
```
# Independence of Errors: Durbin-Watson test statistic
dw_stat = sm.stats.stattools.durbin_watson(residuals)
print("Durbin-Watson statistic:", dw_stat)
# (A value around 2 suggests no autocorrelation.)


# Multicollinearity Analysis: Compute Variance Inflation Factor (VIF)
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train.columns
vif_data["VIF"] = [variance_inflation_factor(X_train.values, i) for i in␣
 ↪range(X_train.shape[1])]
print("\n=== Variance Inflation Factor (VIF) ===")
print(vif_data)
```

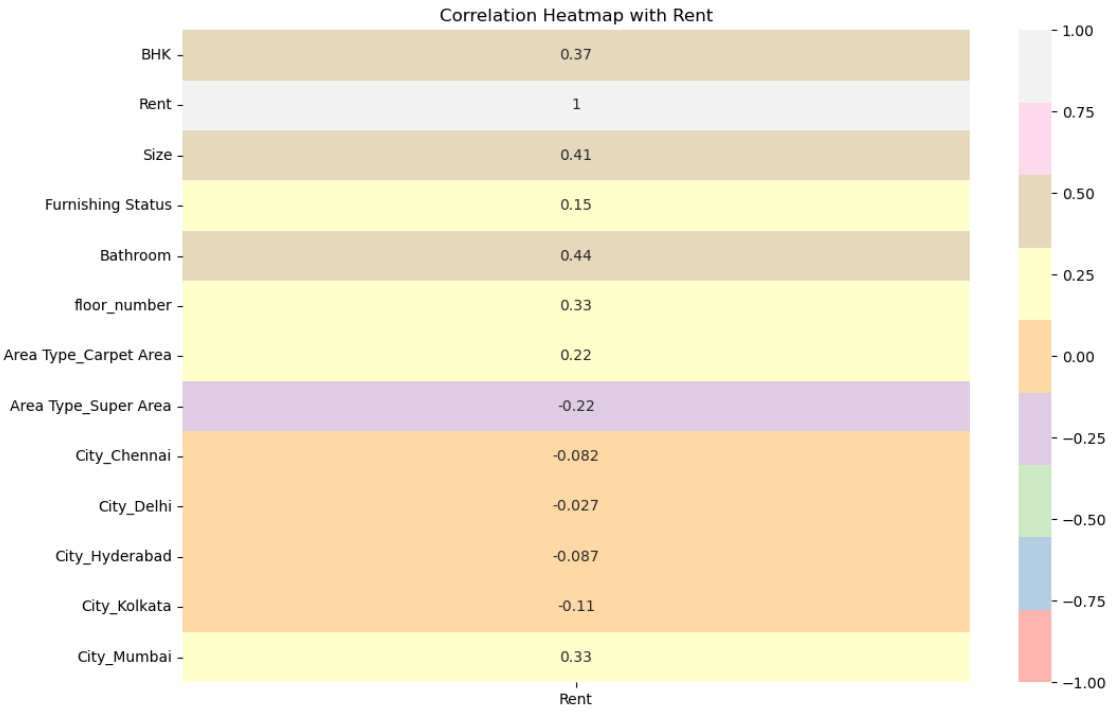
```
Durbin-Watson statistic: 2.0126348081090955

=== Variance Inflation Factor (VIF) ===
                   Feature   VIF
0                      BHK 23.18
1                     Size  9.15
2        Furnishing Status  2.37
3                 Bathroom 23.54
4             floor_number  1.99
5   Area Type_Carpet Area  7.17
6    Area Type_Super Area  6.13
7             City_Chennai  2.02
8               City_Delhi  1.76
9           City_Hyderabad  2.00
```

```
10           City_Kolkata  1.67
11            City_Mumbai  2.81
```

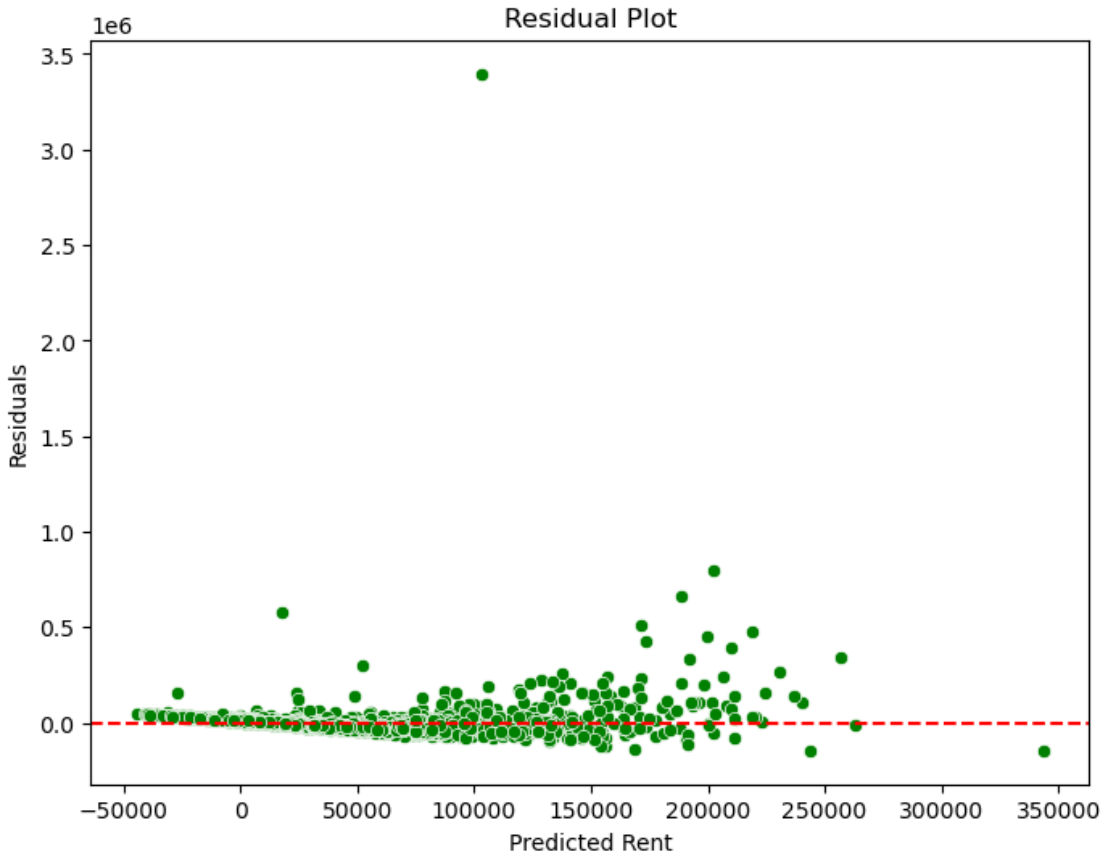
[27]:
```
# Also, a correlation heatmap among independent variables:
plt.figure(figsize=(11, 6))
sns.heatmap(X_train.corr(), annot=True, cmap="Blues")
plt.title("Correlation Heatmap of Independent Variables")
plt.show()
```



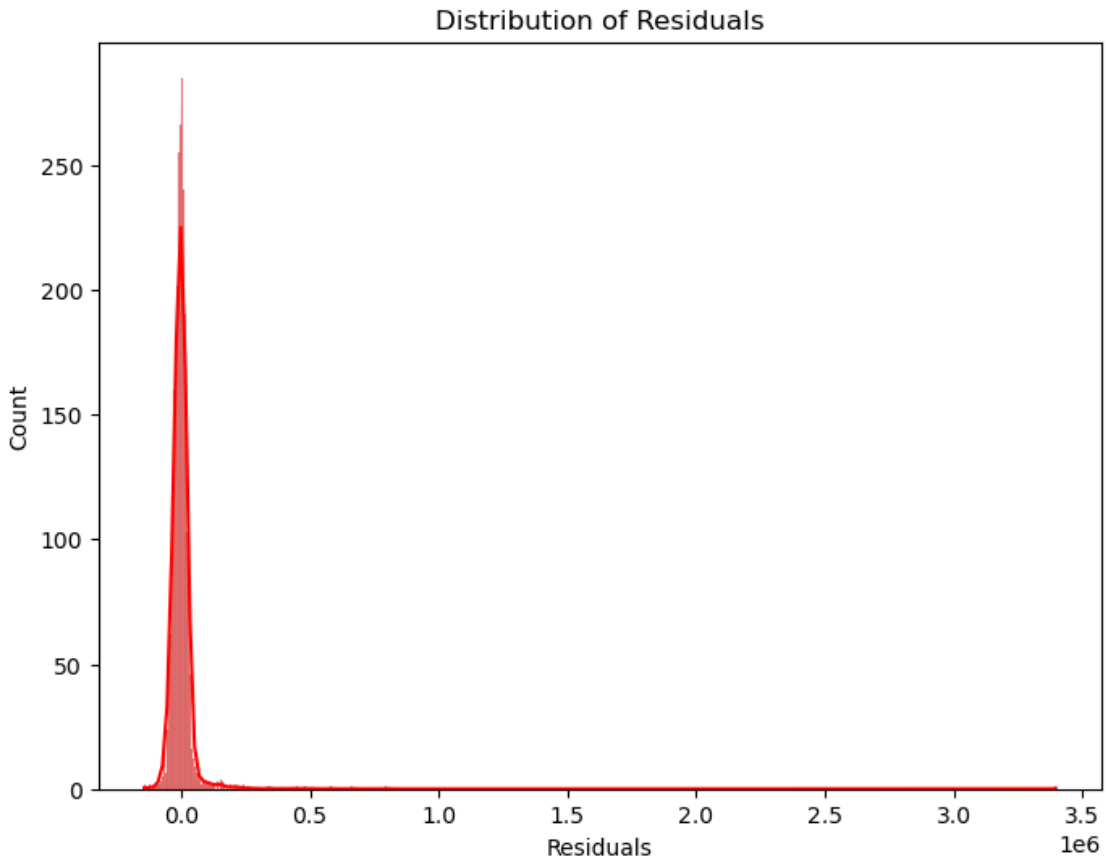
[28]:
```
#HeatMap (Correlation Heatmap with Rent)
plt.figure(figsize=(12, 8))
corr = df_cleaned.corr()
sns.heatmap(corr[['Rent']], annot=True, cmap='Pastel1', vmin=-1, vmax=1)
plt.title('Correlation Heatmap with Rent')
plt.show()
```



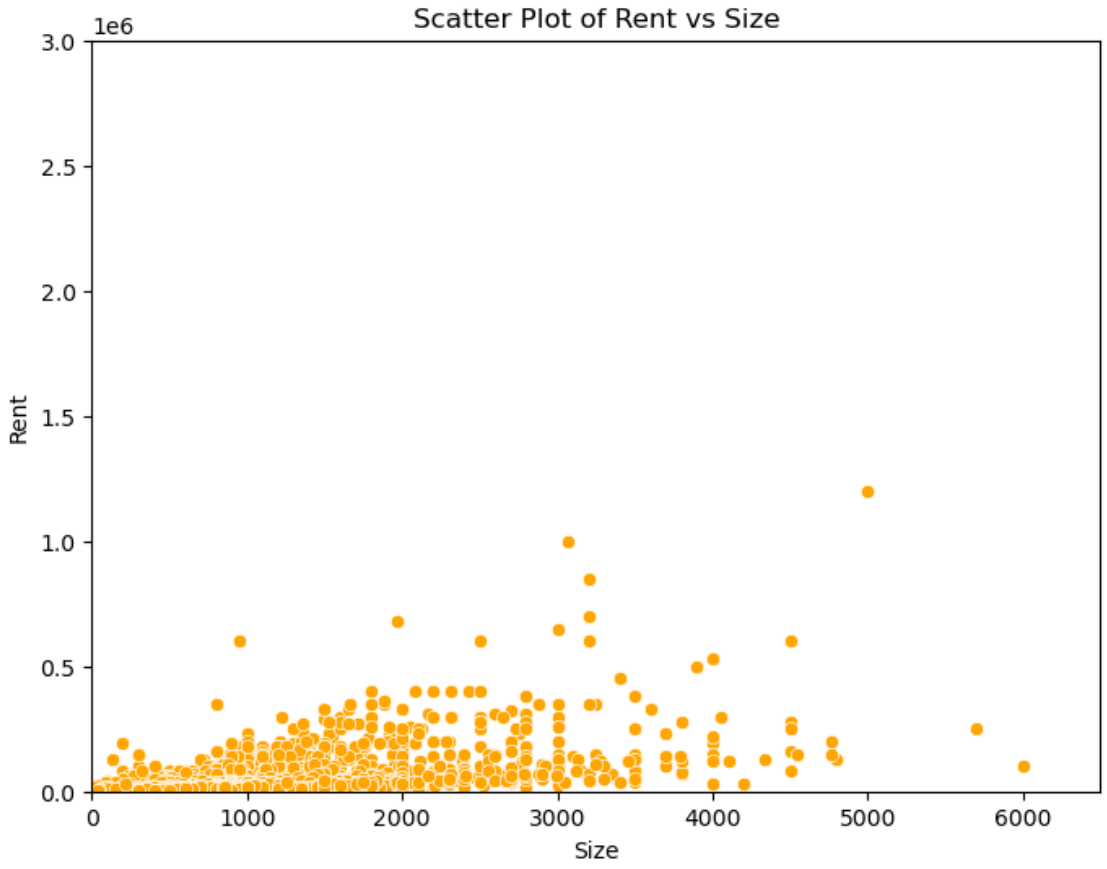
[95]:
```
#ResidualPlot
residuals = y_train - y_pred
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred, y=residuals, color='green')
plt.axhline(y=0, color='red', linestyle='--')
plt.xlabel('Predicted Rent')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.show()
```



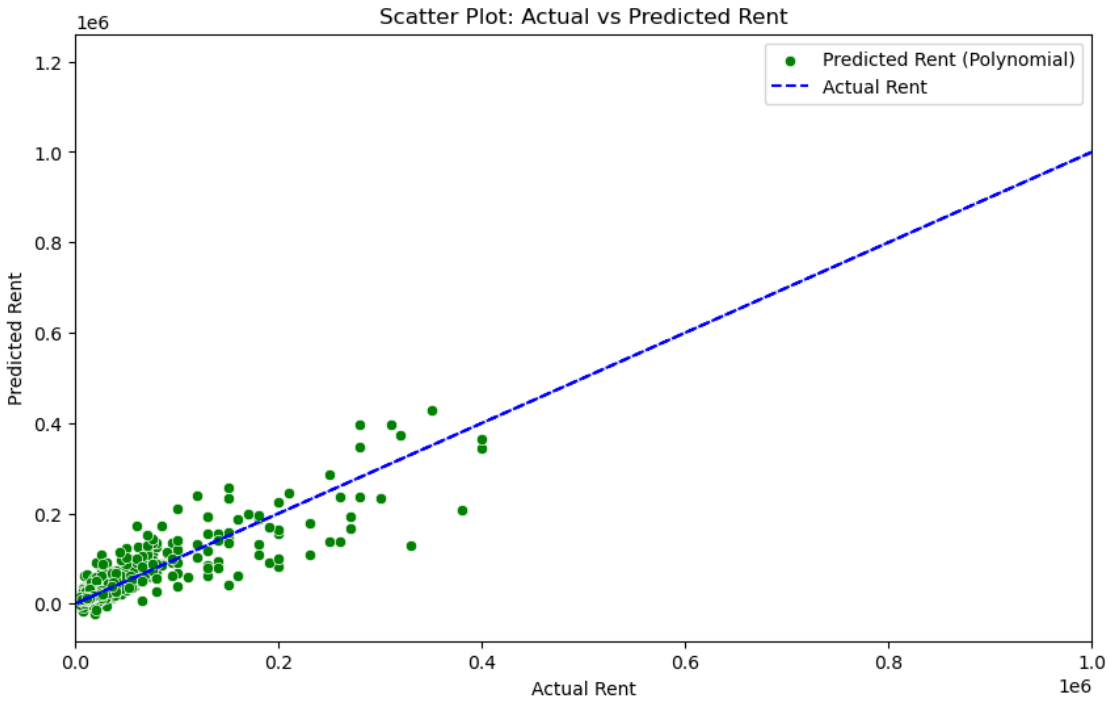
[97]:
```
plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True, color='red')
plt.xlabel('Residuals')
plt.title('Distribution of Residuals')
plt.show()
```



[39]:
```
#scatter plot for rent vs size
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Size', y='Rent', data=df, color='orange')
plt.xlim(0,6500)
plt.ylim(0,3000000)
plt.title('Scatter Plot of Rent vs Size')
plt.xlabel('Size')
plt.ylabel('Rent')
plt.show()
```



[41]:
```
plt.figure(figsize=(10, 6))

# Scatter plot for Polynomial Regression
sns.scatterplot(x=y_test, y=y_poly_pred, color="green", label="Predicted Rent␣
↪(Polynomial)")

# Perfect Fit Line (y = x)
plt.plot(y_test, y_test, color="blue", linestyle="dashed", label="Actual Rent ")
plt.xlim(0,1000000)
# Labels and Title
plt.xlabel("Actual Rent")
plt.ylabel("Predicted Rent")
plt.title("Scatter Plot: Actual vs Predicted Rent")
plt.legend()
plt.show()
```


Scatter Plot: Actual vs Predicted Rent
1e6
Predicted Rent (Polynomial)
Actual Rent
Predicted Rent
Actual Rent
1e6
0.0
0.2
0.4
0.6
0.8
1.0
1.2

[ ]:

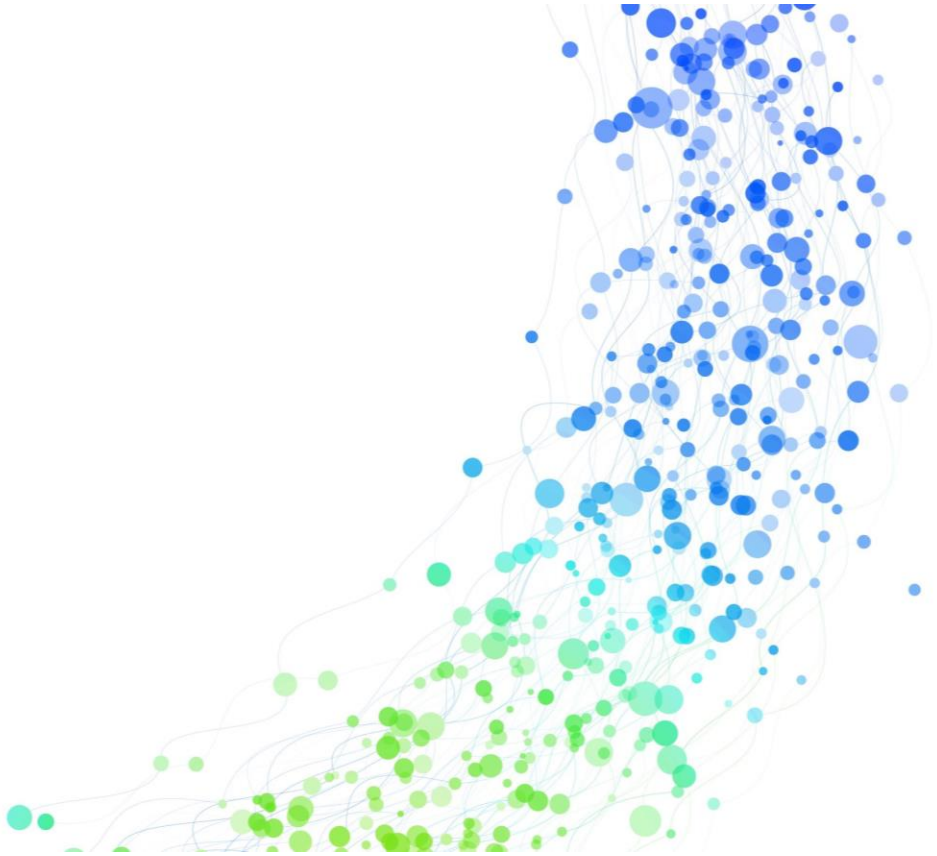
[ ]:

Final Project

SCM 516

by **Synergy Sparks**

- Akanksha Manohar
- Meenakshi Rajeev Nair
- Neha Tiwari
- Niketha Venkatesh
- Shriya Karthikeyan



Link of our Presentation:

- SCM-516_Final_Project

Objective

To harness predictive modeling to extract meaningful insights from data using classification and regression algorithms.

Classification dataset: Naïve Bayes, Decision Tree, Random Forest, KNN, feature importance and conditional probability.

Regression dataset: linear and polynomial regression models to predict the values of a response (target) variable.

By the end, we aim to identify the most effective model and assess its generalization ability.

About Classification Dataset

Customer Churn Prediction Dataset:

Dataset Overview:

- We will be predicting customer churn where definition of the target variable is:
- Churn: 1 = Customer left,
- 0 = Customer stayed.

Key Features:

- Demographics: Age of the customer.
- Subscription Details: Subscription length, contract type (one-year, two-year, month-to-month).
- Usage Behavior: Total data usage (GB), number of support calls.
- Billing & Payment: Payment method, additional services subscribed.

Objective:

- Use machine learning for binary classification to predict churn.
- Help businesses identify at-risk customers and reduce churn.

Preprocessing steps:

- **Dataset Overview:** Collected real-world data with multiple features for classification.
- **Handling Missing Data:** Implemented imputation strategies to ensure data completeness.
- **Label Encoding**: Converted categorical labels into numerical values.
- **Outlier Removal**: Used boxplot analysis to detect and remove outliers.
- **SMOTE Analysis**: Applied SMOTE to balance the dataset.
- **Data Normalization:** Standardized feature values to enhance model efficiency.
- **Splitting Strategy:** Divided data into training and testing sets for evaluation.

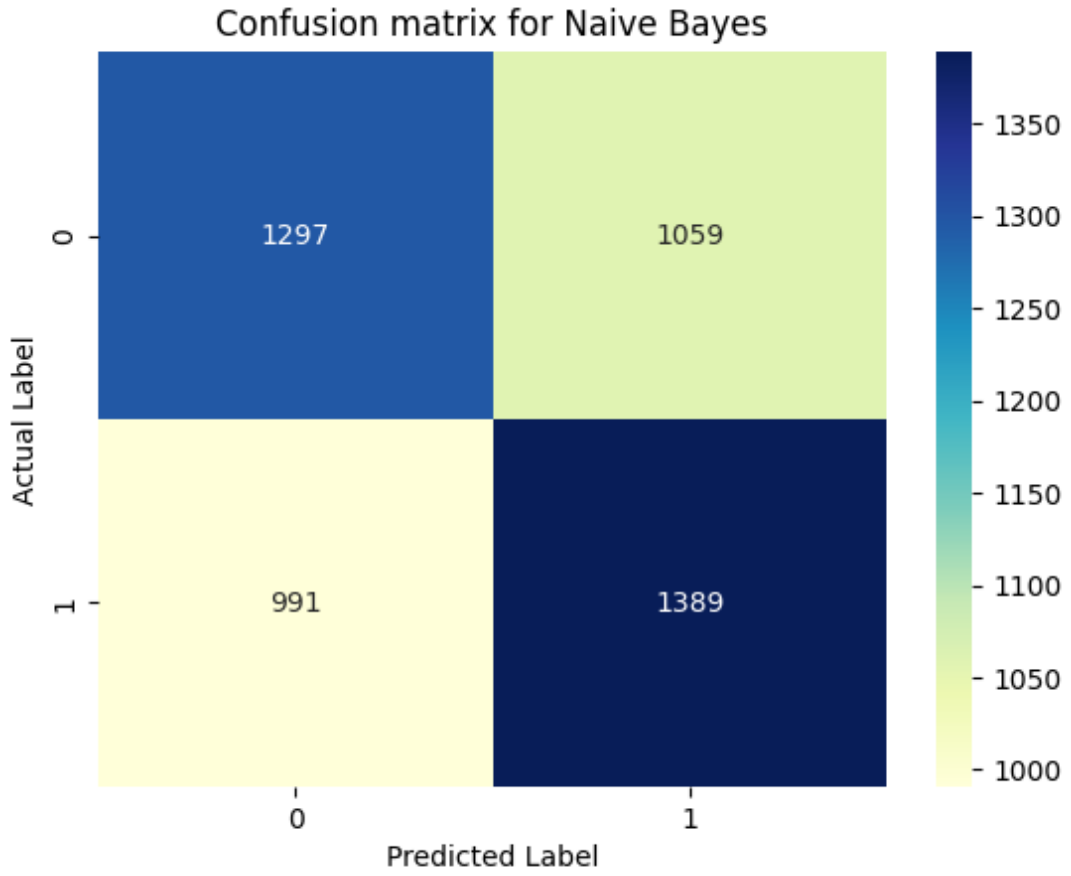
Models

- Naïve Bayes
- Decision Trees
- Random Forest
- KNN

Goodness of Models

- **Confusion Matrix**
- **Accuracy**
- **Precision**
- **Recall**
- **F1 Score**
- **ROC Curve**

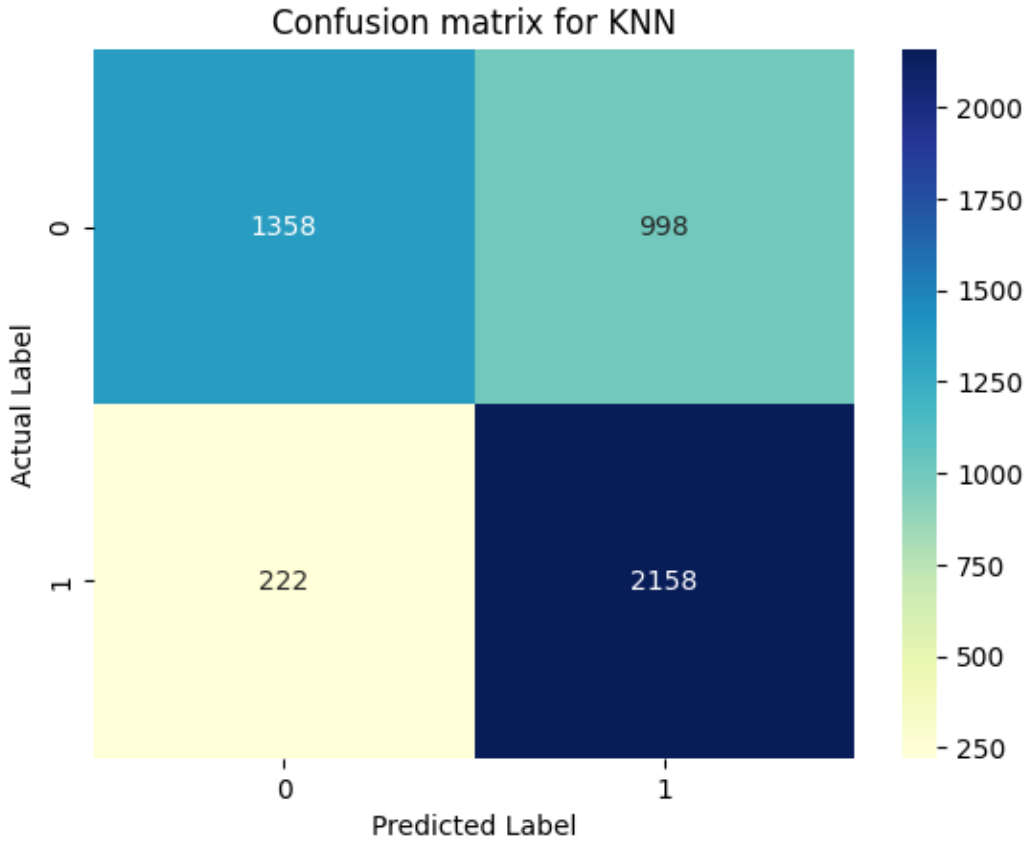
Confusion Matrix for Naïve Bayes



For Naïve Bayes:

Accuracy: 63%
Precision: 62%
Recall: 68%
F1 Score: 65%

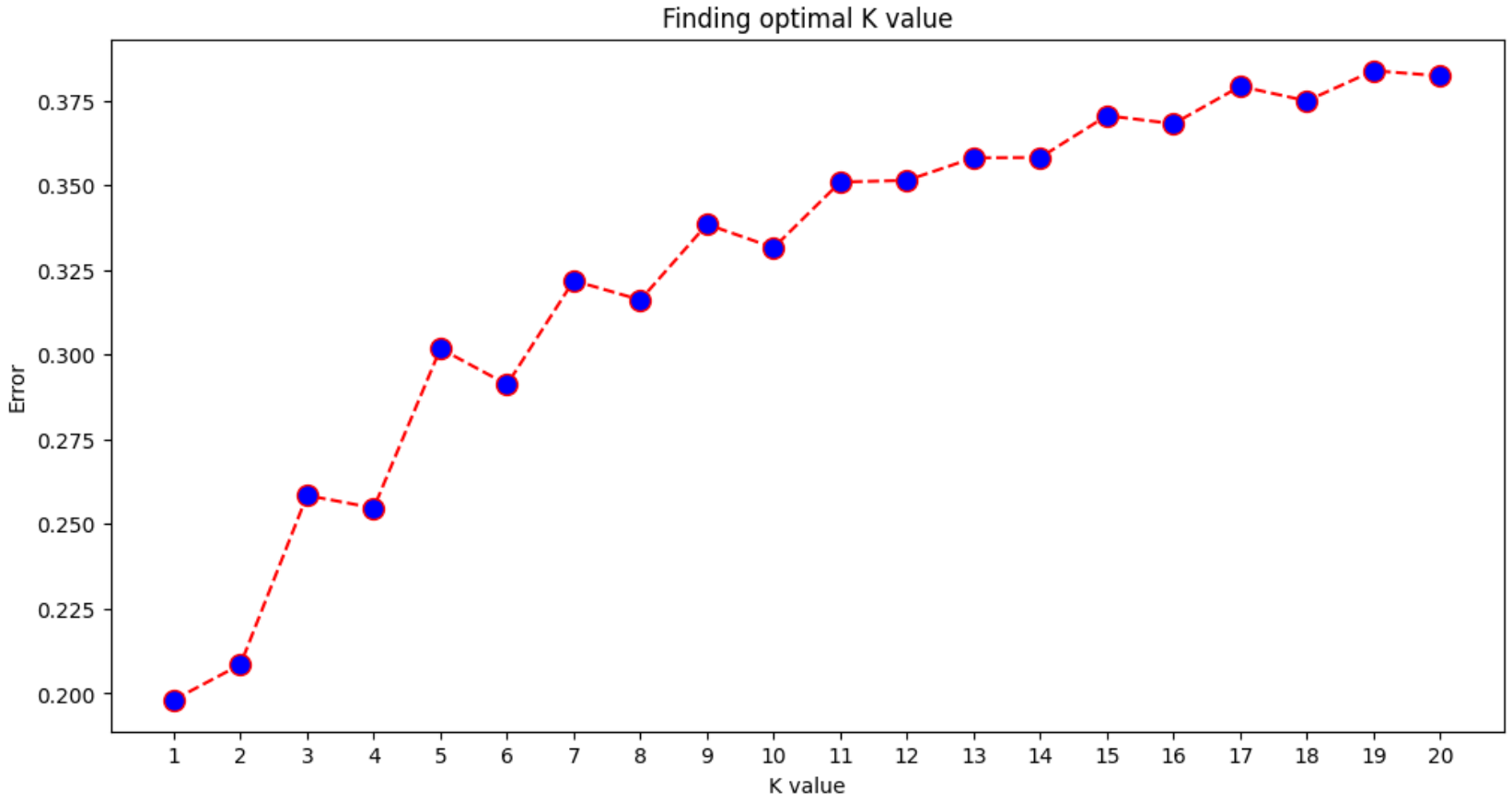
Confusion Matrix for KNN



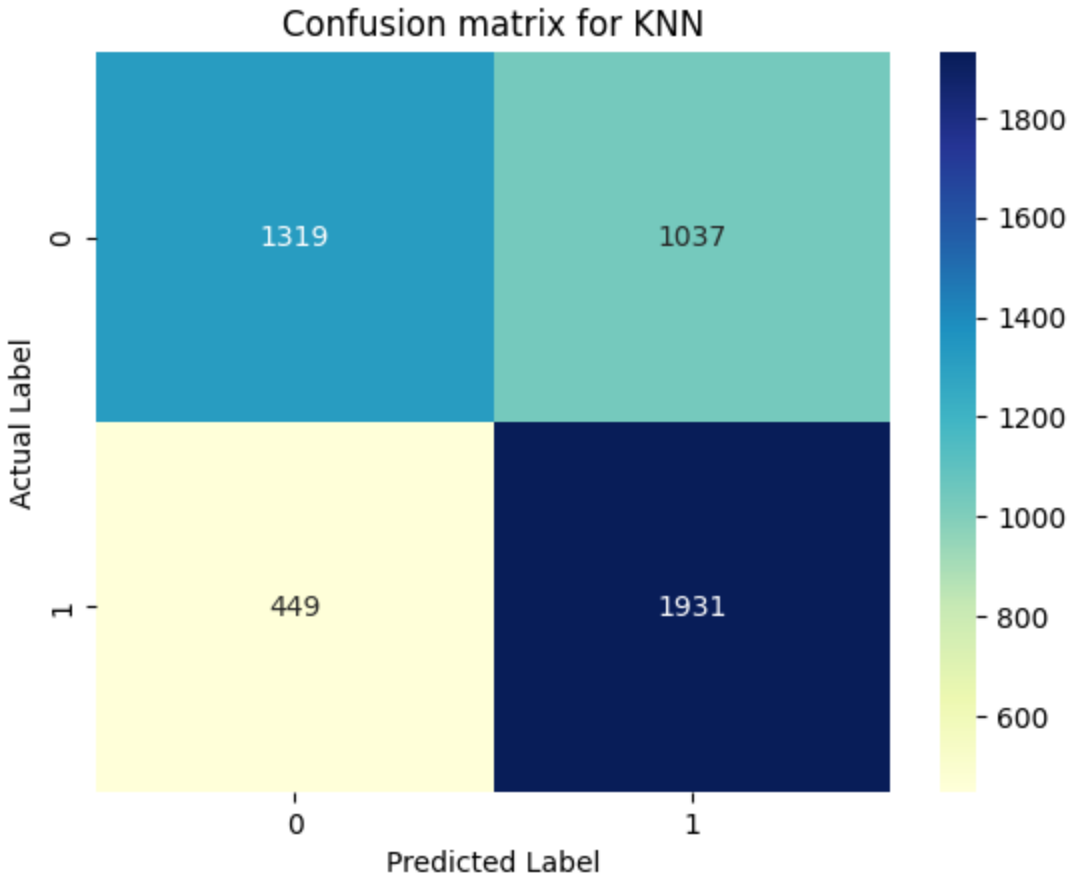
For KNN:

Accuracy: 68%
Precision: 65%
Recall: 78%
F1 Score: 71%

Finding the Optimal K:



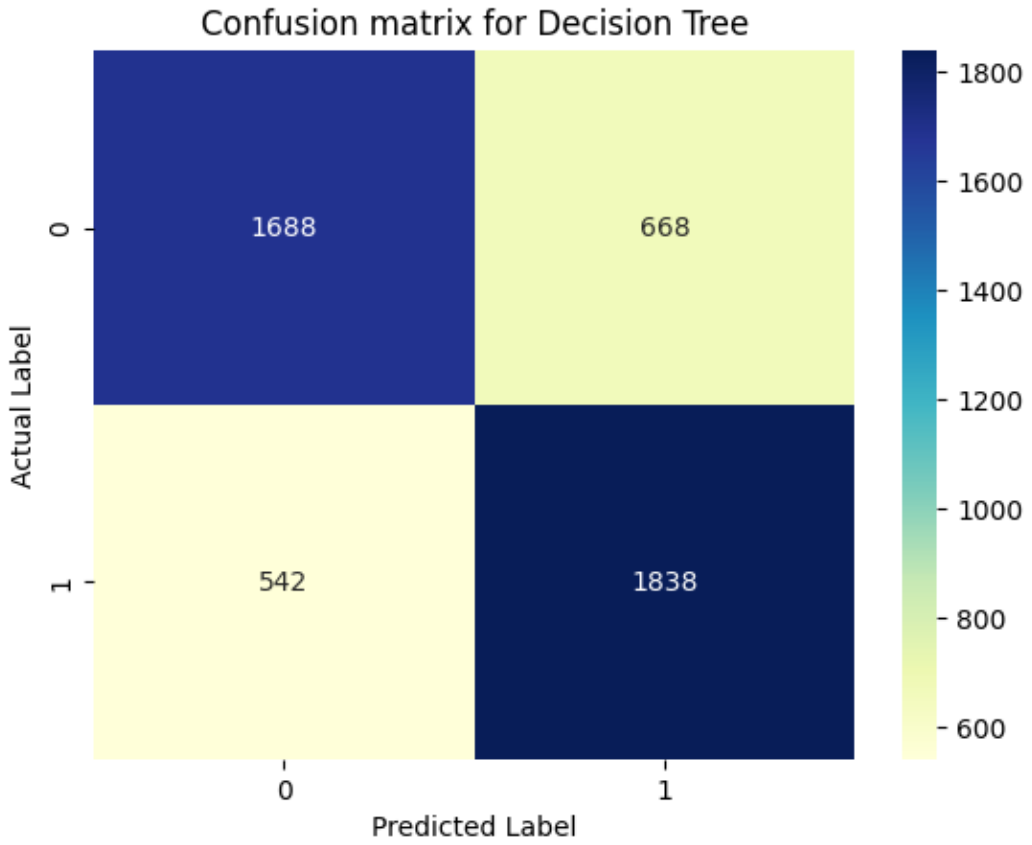
Confusion Matrix for KNN for optimal k=1



For KNN:

Accuracy: 69%
Precision: 65%
Recall: 81%
F1 Score: 72%

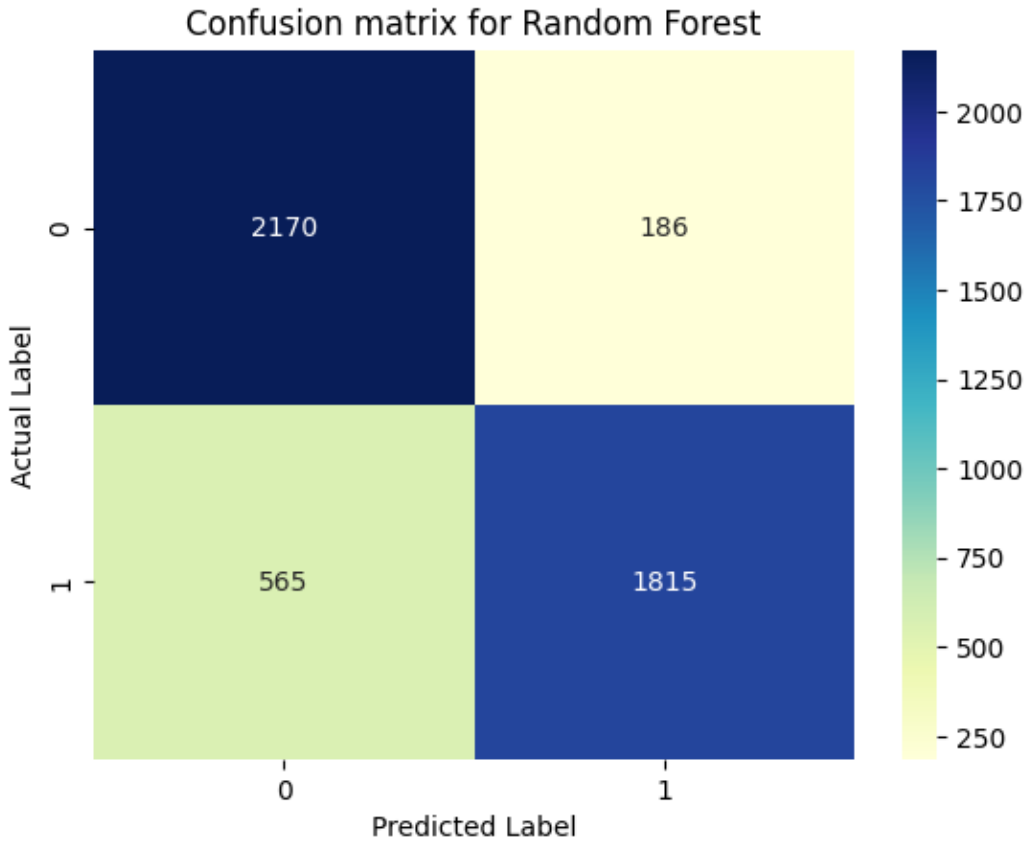
Confusion Matrix for Decision Tree



For Decision Tree:

Accuracy: 76%
Precision: 75%
Recall: 77%
F1 Score: 76%

Confusion Matrix for Random Forest

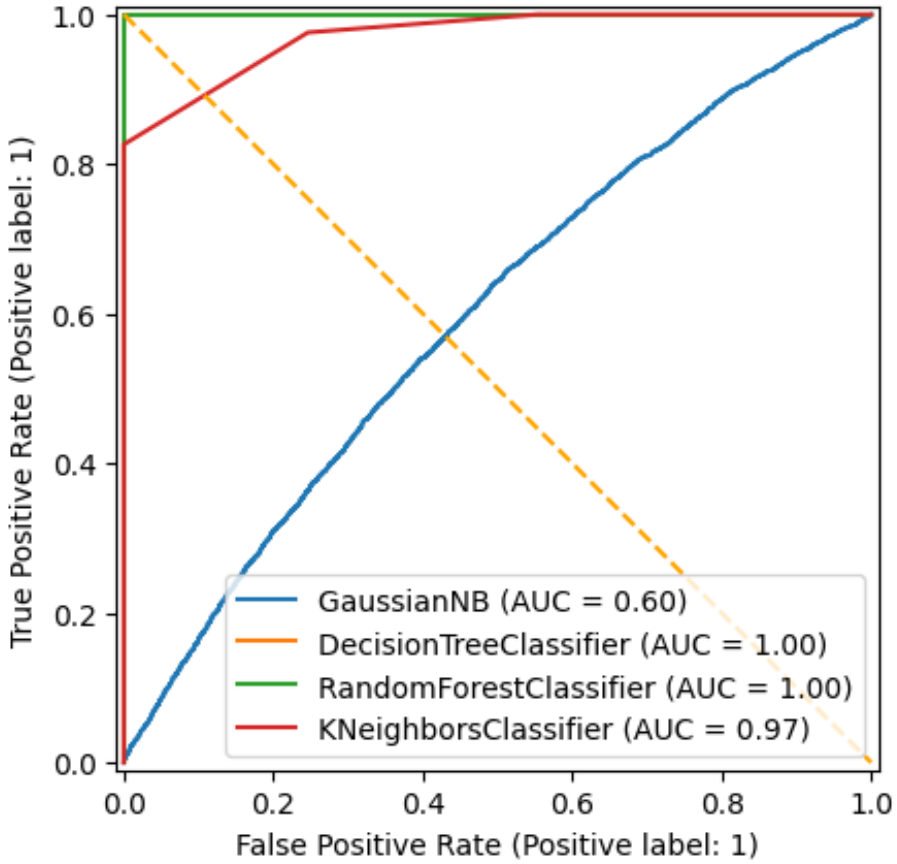


For Random Forest:

Accuracy: 85%
Precision: 97%
Recall: 72%
F1 Score: 83%

ROC Curve Analysis

(for training dataset):



On TRAINING Data-

1. Gaussian NB : Its curve is close to the diagonal line, indicating it's barely better than random guessing. The AUC is also relatively low.

2. K neighbors Classifier : This model performs very well, with its curve close to the top-left corner.

3.Decision Tree Classifier : This model has perfect performance. However, a perfect AUC often suggests potential overfitting.

4.Random Forest Classifier : Similar to the Decision Tree, the same caution about potential overfitting applies.

ROC Curve Analysis

(for testing dataset):

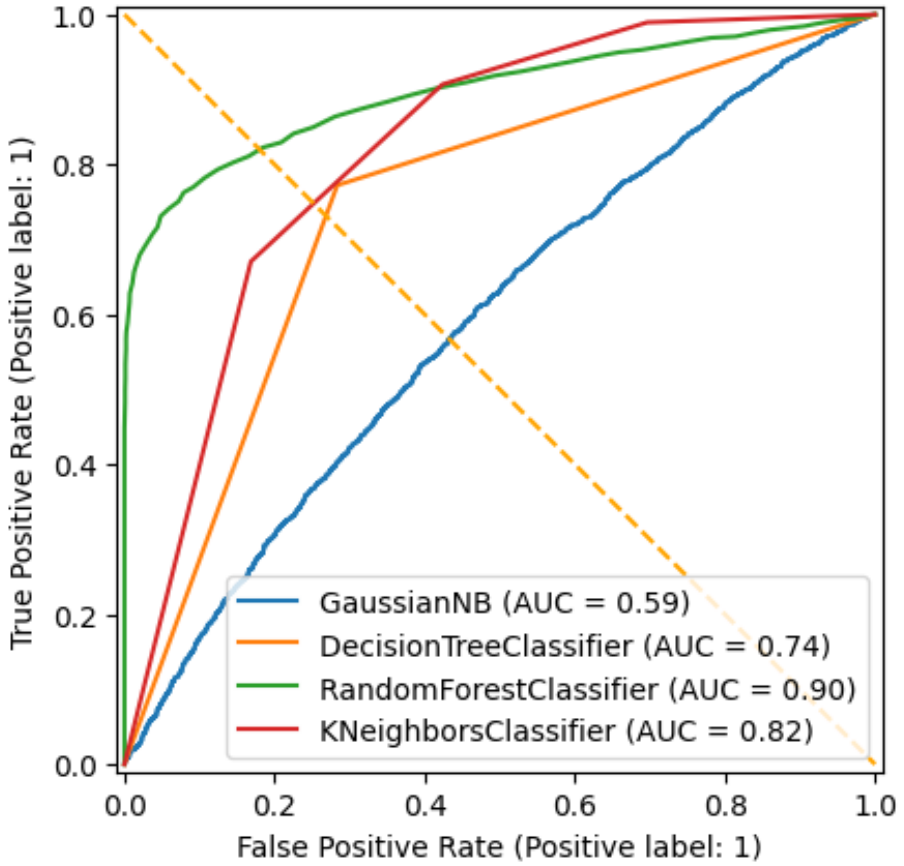
On TESTING DATA-

1. **GaussianNB (AUC = 0.59):** Very poor. The curve is close to the diagonal line.

2. **Kneighbors Classifier (AUC = 0.82):** Reasonably good performance. The curve is above the diagonal line. This model also performs well on the testing data, though not as strongly as Random Forest.

3. **Decision Tree Classifier (AUC = 0.74):** Moderate performance. The curve is above the diagonal line, but not significantly.

4. **Random Forest Classifier (AUC = 0.90):** Good performance. The curve is well above the diagonal line, indicating a strong ability to distinguish between classes. **This model generalizes well to the testing data and is a good choice for this classification task.**



Performance Comparison

Evaluating Model Effectiveness

- **Accuracy:** Random Forest achieved the highest accuracy, outperforming other models.
- **Precision & Recall:** Trade-off observed: Decision Trees had high precision but lower recall, while KNN struggled with both.
- **F1-Score:** Random Forest maintained the best balance.
- **Class Imbalance:** Some models were impacted by imbalanced data, affecting recall and precision.

Overall :

Random Forest demonstrated the best overall performance across all metrics.

Classification report of Random Forest

	Precision	Recall	F1- score	support
Class 0	0.79	0.92	0.85	2356
Class 1	0.91	0.76	0.83	2380
Accuracy			0.84	4736
Macro avg	0.85	0.84	0.84	4736
Weighted avg	0.85	0.84	0.84	4736

Conclusion: The model has an overall accuracy of 84% (0.84).

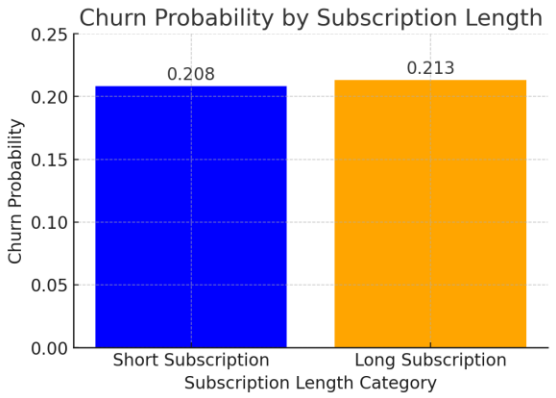
This means it correctly predicted the class labels for 84% of the samples in the test set.

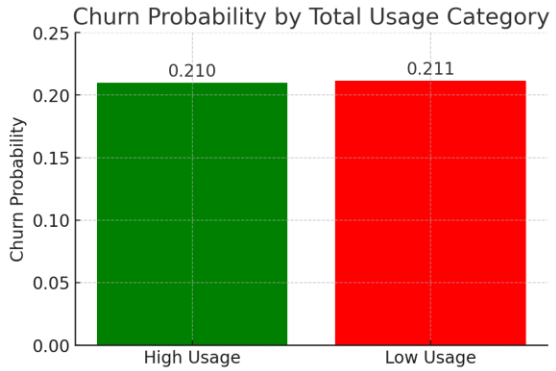

Let's see how
well our model
performs on
unseen data?

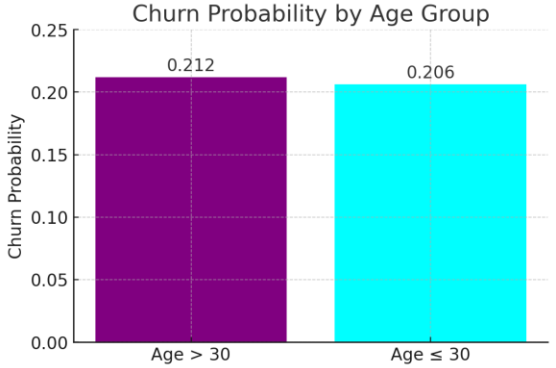
New dataset:

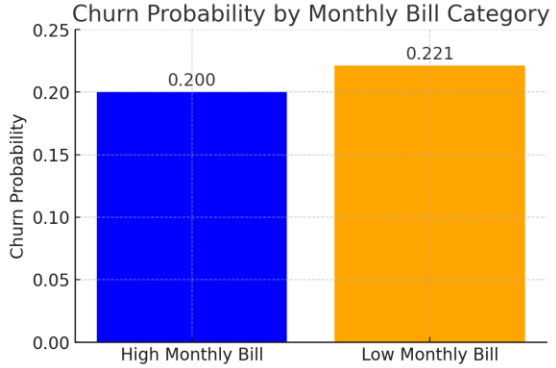
	Customer ID	Age	Subscription Length (months)	Monthly Bill ($)	Total Usage (GB)	Support Calls	Contract Type	Payment Method	Has Additional Services
1	10001	43.296	33.66	107.8	230.95	1	Month-to-Month	Debit Card	1
2	10002	55.704	48.45	144.6	398.05	9	One Year	Credit Card	1
3	10003	35.928	27.08	71.65	131.65	1	Month-to-Month	PayPal	0

Feature Importance using Naïve Bayes



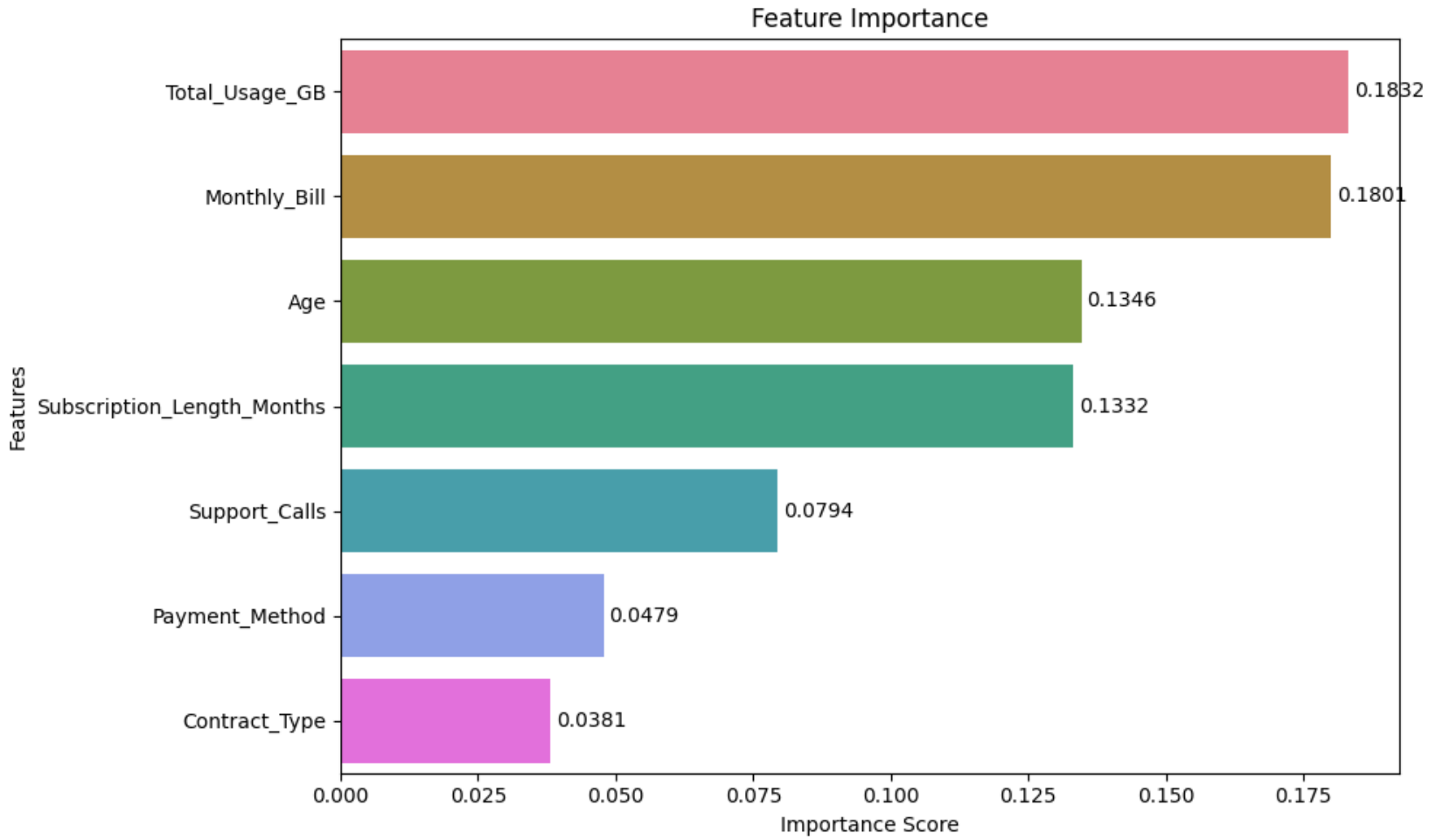






Importance of Feature

(determined using Random Forest):



Which model performed best?

- **Random Forest** emerged as the **top-performing model** due to its ensemble learning approach. Unlike a single decision tree, it aggregates predictions from multiple trees, significantly improving accuracy.
- One of its key strengths is robustness—Random Forest is less sensitive to noisy data and missing values compared to other classifiers.
- It also provides feature importance rankings, helping us understand which variables influence predictions the most.
- Moreover, it strikes an excellent balance between bias and variance, avoiding overfitting while maintaining high predictive power.
- Our results confirm that **Random Forest** consistently outperformed other models across all key performance metrics.

New dataset with churn values:

	Customer ID	Age	Subscription Length (months)	Monthly Bill ($)	Total Usage (GB)	Support Calls	Contract Type	Payment Method	Has Additional Services
1	10001	43.296	33.66	107.8	230.95	1	Month-to-Month	Debit Card	1
2	10002	55.704	48.45	144.6	398.05	9	One Year	Credit Card	1
3	10003	35.928	27.08	71.65	131.65	1	Month-to-Month	PayPal	0

No Churn	Churn = 1
62%	38%
72%	28%
61%	39%

Are there any signs of overfitting?

While machine learning models offer powerful predictive capabilities, they come with challenges and limitations.

- **Data quality** —no model can outperform the quality of its input data.
- **Overfitting** – where a model performs exceptionally well on training data but struggles on new data.
- **Computational cost** –models like Random Forest require significant processing power, making them less feasible for real-time applications.
- **Hyperparameter tuning** – finding the right balance between bias and variance.
- **Interpretability** – making it difficult to explain predictions, especially in fields requiring transparency, such as healthcare and finance.

Key Insights from Conditional Probability Analysis of Customer Churn:

Churn Risk Factors:

- Customers with **high monthly bills ($90+)**, **heavy data usage (300GB+)**, or **multiple support calls (5+ calls)** → (~20%)
- Month-to-Month contract customers (especially heavy users) are at **higher churn risk (33%)**
- Customers with **long-term contracts (2+ years, 36+ months)** have slightly lower churn (~21%)
- **Older customers (60+) with low usage (<100GB)** have the lowest churn probability (~18%)

Regression Analysis:

About Regression Dataset

House Rent Prediction Dataset:2

Dataset Overview:

- Target variable is rent.
- Aims to predict rent prices based on other independent variables.

Key Features:

- Includes numerical attributes like BHK, size, bathroom.
- Includes categorical attributes like floor level, city, furnishing status and tenant preferences.

Objective:

- Apply Multiple Linear and Polynomial Regression to predict rental prices based on relevant features.
- Analyze the impact of different property attributes on rent to improve decision-making for renters and property owners.

Independent variables:

	BHK	Size	Furnishing status	Bathroom	Floor number	Carpet Area	Super Area	Chennai	Delhi	Hyderabad	Kolkata	Mumbai
0	2	1100	0	2	0	False	True	False	False	False	True	False
1	2	800	1	1	1	False	True	False	False	False	True	False
2	2	1000	1	1	1	False	True	False	False	False	True	False
3	2	800	0	1	1	False	True	False	False	False	True	False
4	2	850	0	1	1	True	False	False	False	False	True	False

Preprocessing steps:

- **Dataset Overview:** Real-world housing rental data with features like rent, size, location, and furnishing status.
- **Handling Missing Data:** Used imputation techniques to ensure data completeness.
- **Categorical to Numerical Conversion**: Transformed categorical features for model compatibility.
- **One-Hot Encoding**: Applied one-hot Encoding to multi-category variables to retain information.
- **Boolean Conversion**: Converted Boolean columns to integers (0/1).
- **Splitting Strategy:** Split data into training and testing sets for evaluation.

Equation of the Model:

```
=== Equation of the Model ===
Rent = 236693.67 + -40788374731.34 * BHK + 81575716.39 * Size + 174315.13 * Furnishing Status + 157807.45 * Bathroom + 26404.82 * floor_numbe
r + -112617.09 * Area Type_Carpet Area + -114163.93 * Area Type_Super Area + 52728.54 * City_Chennai + -4197.02 * City_Delhi + -4768.28 * Cit
y_Hyderabad + 509.40 * City_Kolkata + -8347.23 * City_Mumbai + -382.45 * BHK^2 + -4.24 * BHK Size + -1763.72 * BHK Furnishing Status + 6712.4
3 * BHK Bathroom + 891.13 * BHK floor_number + 40788361229.53 * BHK Area Type_Carpet Area + 40788362939.56 * BHK Area Type_Super Area + 8132.
09 * BHK City_Chennai + 3667.74 * BHK City_Delhi + 7881.57 * BHK City_Hyderabad + 10132.94 * BHK City_Kolkata + -20261.69 * BHK City_Mumbai +
0.00 * Size^2 + 7.13 * Size Furnishing Status + 4.03 * Size Bathroom + -1.07 * Size floor_number + -81575669.32 * Size Area Type_Carpet Area
+ -81575685.43 * Size Area Type_Super Area + -33.46 * Size City_Chennai + -34.51 * Size City_Delhi + -38.79 * Size City_Hyderabad + -42.21 *
Size City_Kolkata + 109.30 * Size City_Mumbai + -682.73 * Furnishing Status^2 + -736.49 * Furnishing Status Bathroom + 648.78 * Furnishing St
atus floor_number + -175857.84 * Furnishing Status Area Type_Carpet Area + -175173.56 * Furnishing Status Area Type_Super Area + 1054.59 * Fu
rnishing Status City_Chennai + 1934.58 * Furnishing Status City_Delhi + -679.30 * Furnishing Status City_Hyderabad + 1298.05 * Furnishing Sta
tus City_Kolkata + 6327.87 * Furnishing Status City_Mumbai + -3317.53 * Bathroom^2 + 141.87 * Bathroom floor_number + -165789.41 * Bathroom A
rea Type_Carpet Area + -165857.30 * Bathroom Area Type_Super Area + 4721.69 * Bathroom City_Chennai + 22330.81 * Bathroom City_Delhi + 1478.5
8 * Bathroom City_Hyderabad + 5507.58 * Bathroom City_Kolkata + 4141.27 * Bathroom City_Mumbai + -14.38 * floor_number^2 + -28583.44 * floor_
number Area Type_Carpet Area + -28223.98 * floor_number Area Type_Super Area + 768.83 * floor_number City_Chennai + -360.44 * floor_number Ci
ty_Delhi + 1869.63 * floor_number City_Hyderabad + -113.98 * floor_number City_Kolkata + 1164.90 * floor_number City_Mumbai + -112617.06 * Ar
ea Type_Carpet Area^2 + 0.00 * Area Type_Carpet Area Area Type_Super Area + -109708.23 * Area Type_Carpet Area City_Chennai + -8605.85 * Area
Type_Carpet Area City_Delhi + 7524.22 * Area Type_Carpet Area City_Hyderabad + -4002.08 * Area Type_Carpet Area City_Kolkata + 1563.71 * Area
Type_Carpet Area City_Mumbai + -114163.93 * Area Type_Super Area^2 + -100236.42 * Area Type_Super Area City_Chennai + 4408.78 * Area Type_Sup
er Area City_Delhi + 23599.69 * Area Type_Super Area City_Hyderabad + 4511.47 * Area Type_Super Area City_Kolkata + -9910.94 * Area Type_Supe
r Area City_Mumbai + 52728.53 * City_Chennai^2 + 0.00 * City_Chennai City_Delhi + 0.00 * City_Chennai City_Hyderabad + 0.00 * City_Chennai Ci
ty_Kolkata + 0.00 * City_Chennai City_Mumbai + -4197.07 * City_Delhi^2 + 0.00 * City_Delhi City_Hyderabad + 0.00 * City_Delhi City_Kolkata +
0.00 * City_Delhi City_Mumbai + -4768.27 * City_Hyderabad^2 + 0.00 * City_Hyderabad City_Kolkata + 0.00 * City_Hyderabad City_Mumbai + 509.39
* City_Kolkata^2 + 0.00 * City_Kolkata City_Mumbai + -8347.24 * City_Mumbai^2
```

Interpretation of the Regression Equation

Rent is influenced by multiple factors, including:

- **Size**: Larger properties generally have higher rents.
- **BHK**: Unexpected large negative impact suggests possible data issues.
- **Furnishing Status**: Furnished properties command higher rents.
- **Bathrooms**: More bathrooms increase rent.
- **Floor Number**: Higher floors slightly increase rent.

Regression model methods:

- Multiple Linear Regression
- Polynomial Regression

Interpretation of R_2 and adjusted R_2 (using Linear & Polynomial Regression):

- For Multiple Linear Regression:
  - **R2:** 0.522884332926580 7
  - **Adjusted R2:** 0.516773993540368 2

For Polynomial Regression (degree=2):

R2: 0.8137733813128744

Adjusted R2: 0.7942618613107308

Interpretation of R2 & R2 adjusted values:

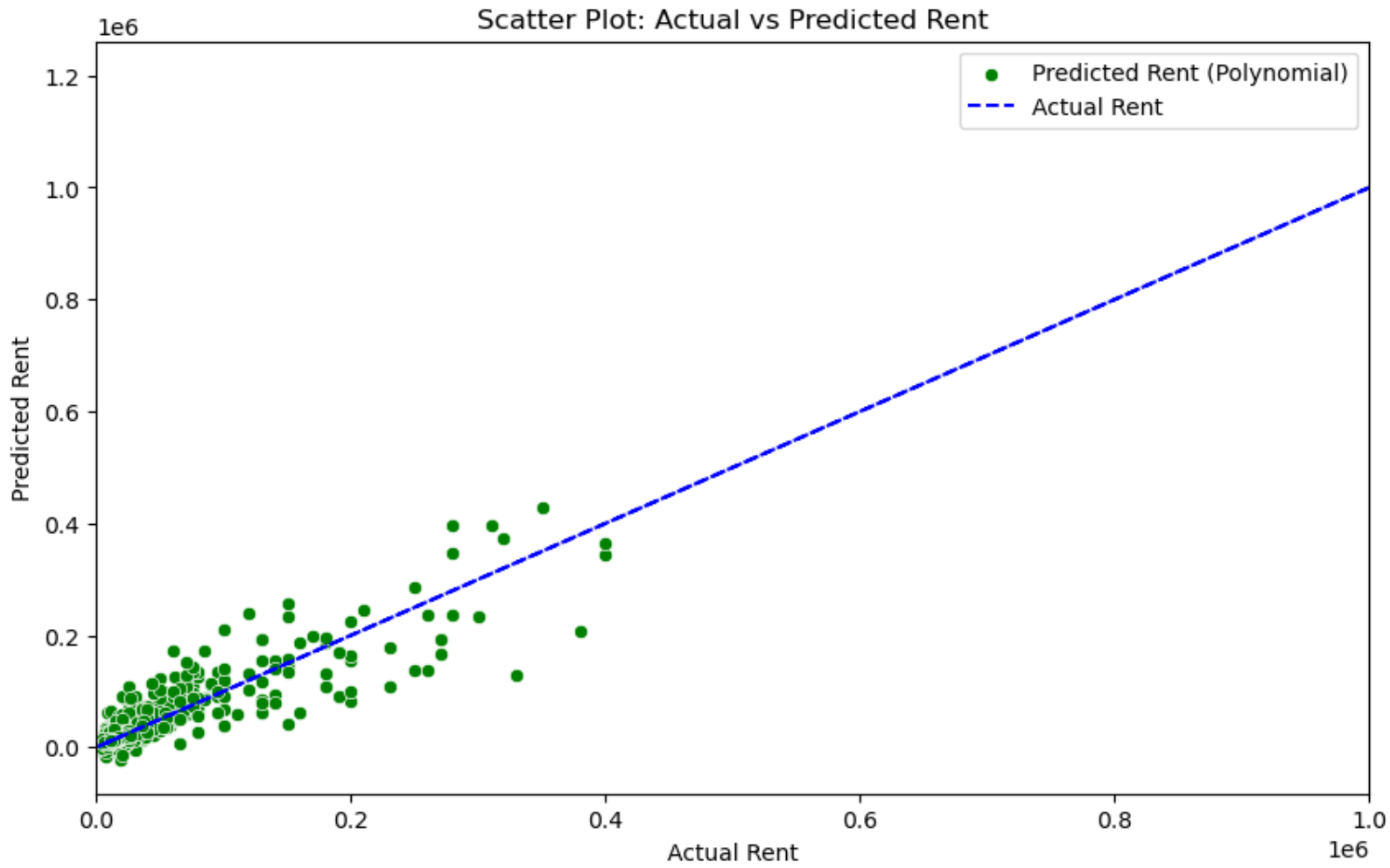
- **Multiple Linear Regression**:
  - **R2 = 0.52**→ The model explains 52.29% of the variance in rent prices can be explained by the independent features.
  - **Adjusted R2** → After accounting for the number of predictors, the model still explains **51.7%** of the variance, confirming that adding more features did not drastically improve the fit.

- **Polynomial Regression:**
  - **R2 = 0.81** → The model explains 81.38% of the variance in rent prices can be explained bu the independent variable , which is significantly higher than the multiple linear regression model.
  - **Adjusted R^2 = 0.51** →After adjusting for additional predictors, the model still explains **79.4%** of the variance.

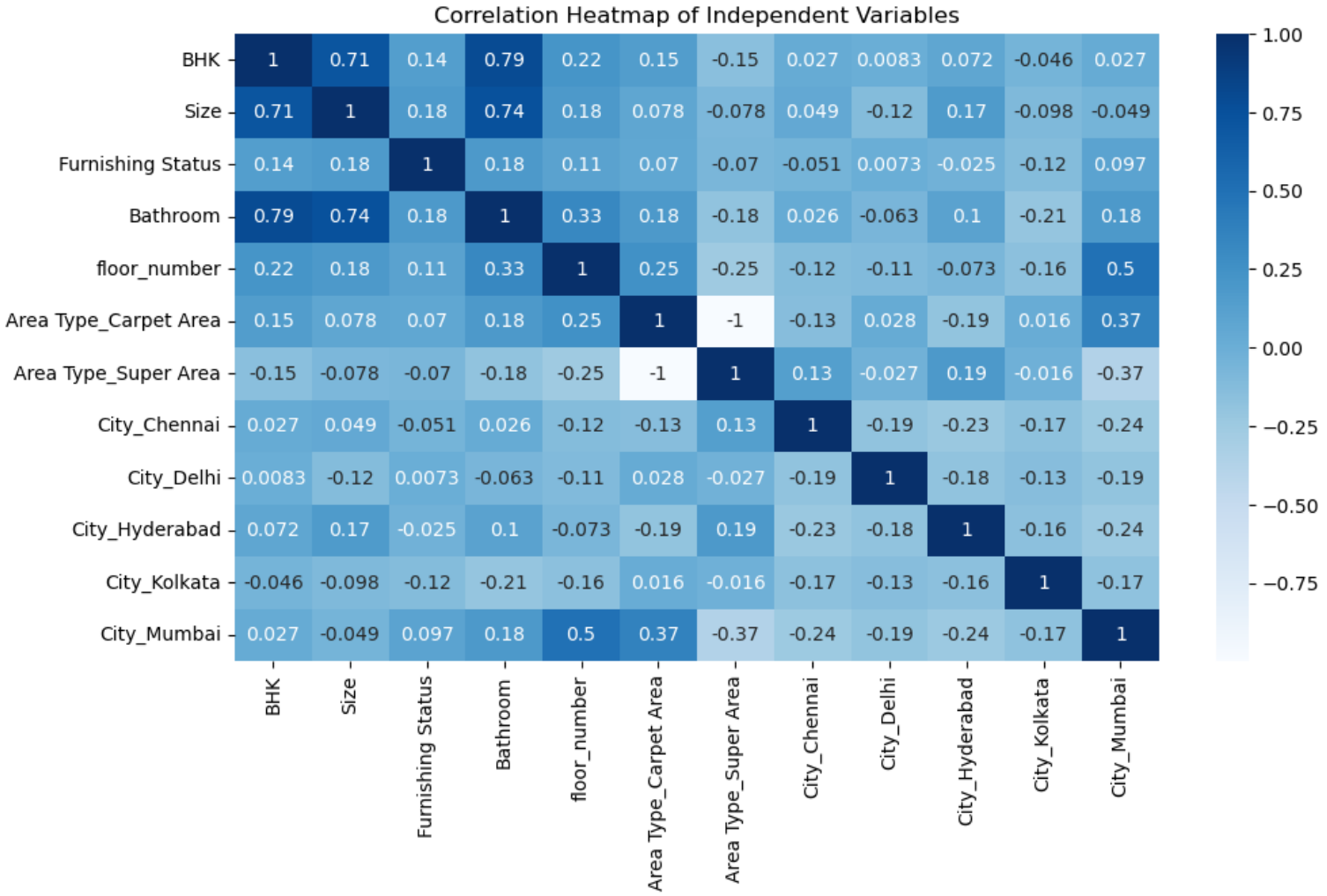
INTERPRETATION:

Polynomial regression outperforms multiple linear regression in explaining the data, indicating that a nonlinear model is more suitable for our dataset.

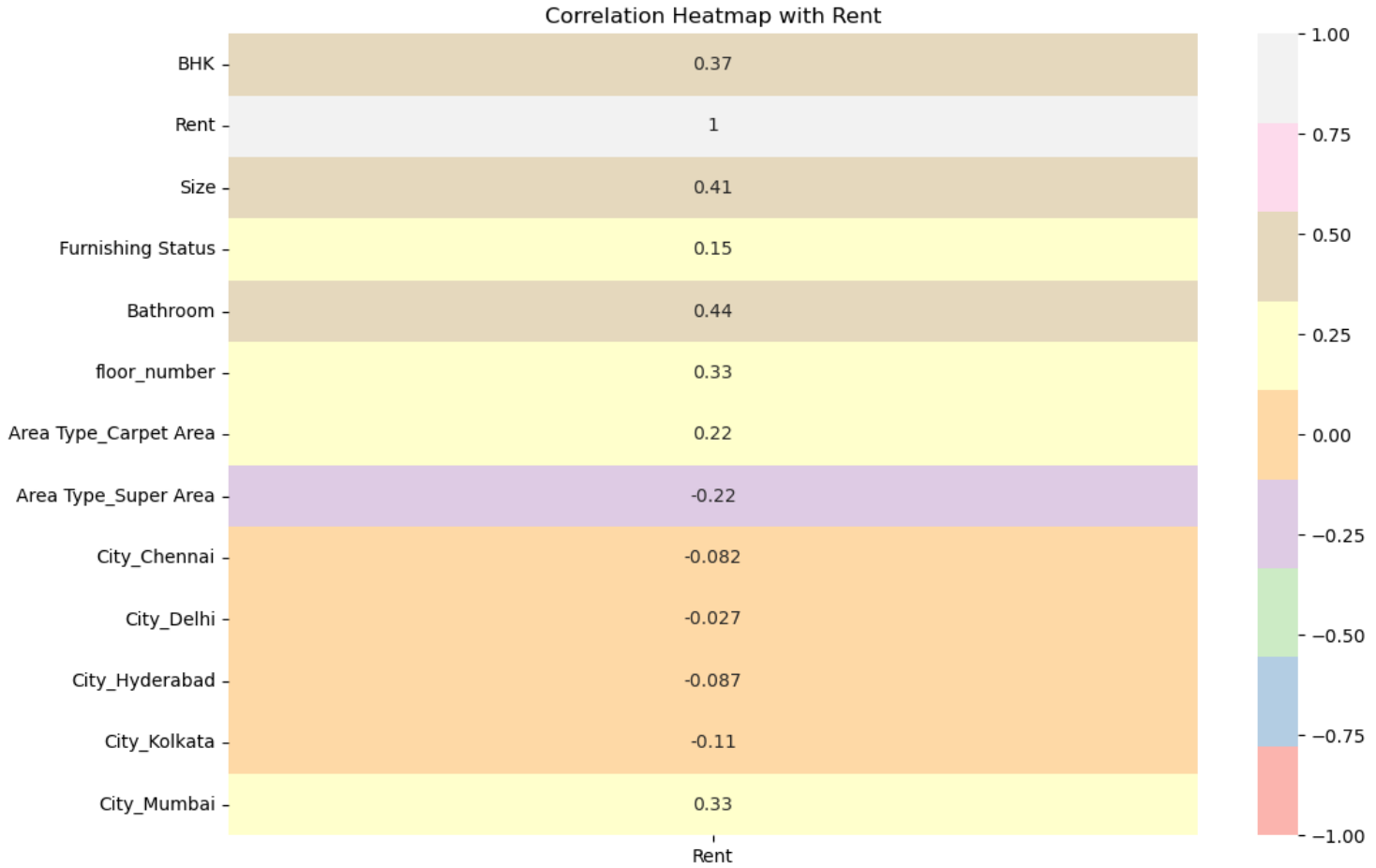
Actual vs Predicted Rent:



Correlation between Input variables:

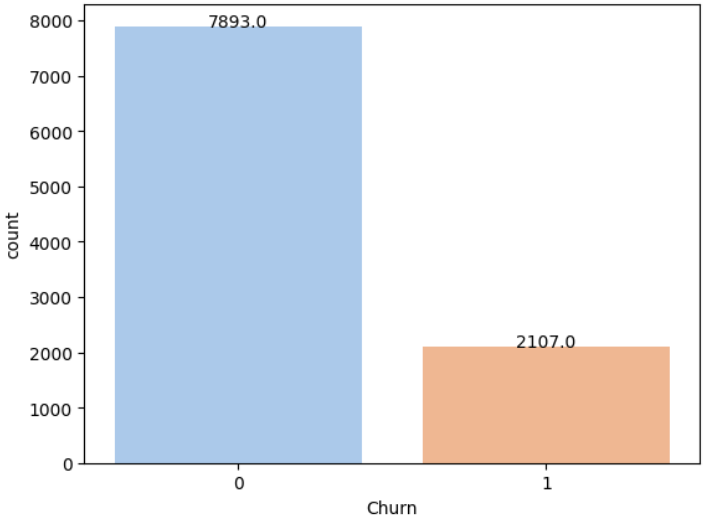


Correlation between Input & Output variable:

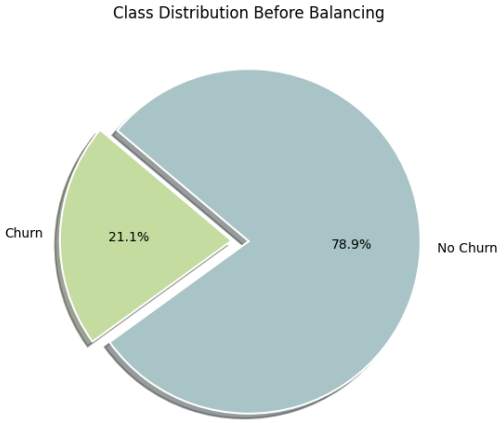


Visualization

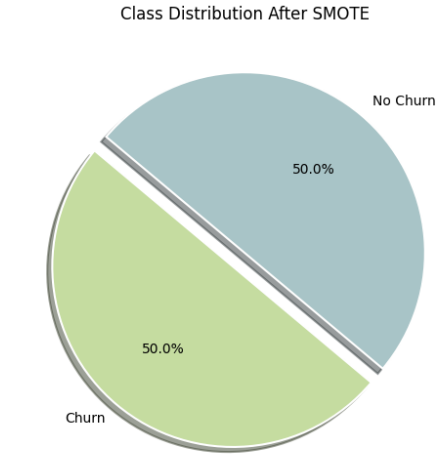
Visualizations for Binary Classification Analysis:



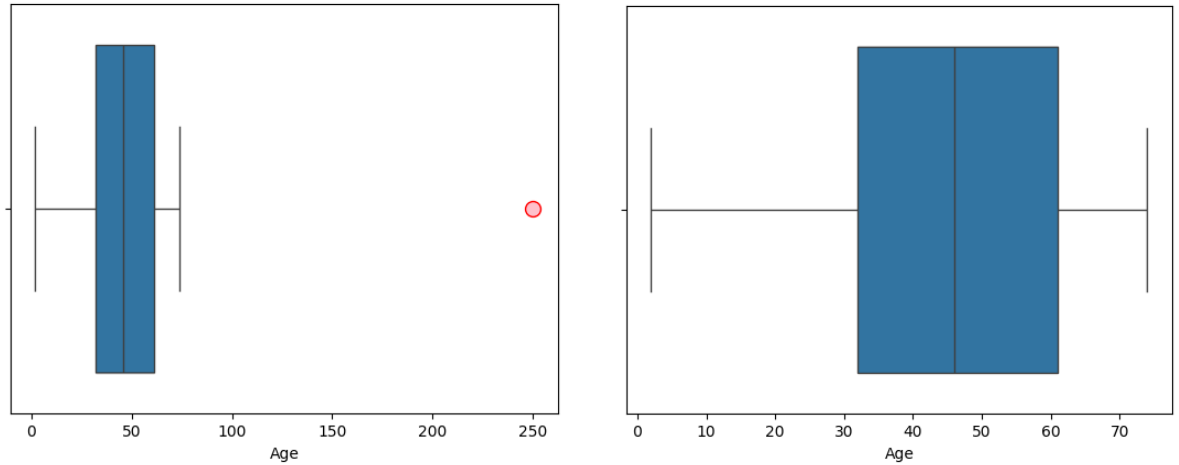
Imbalanced data



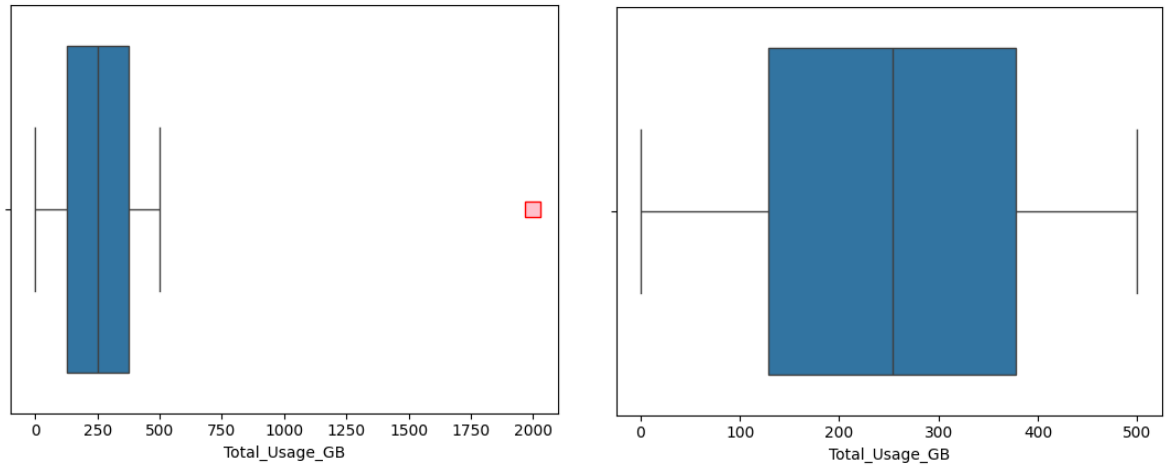
Class Distribution Before Balancing



Class Distribution After SMOTE

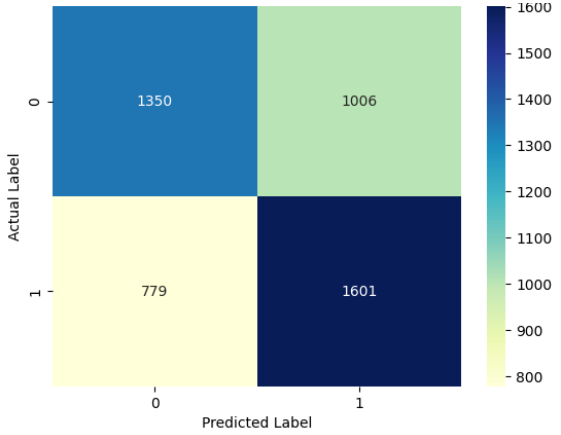


Outliers detected and removed for “Age” column

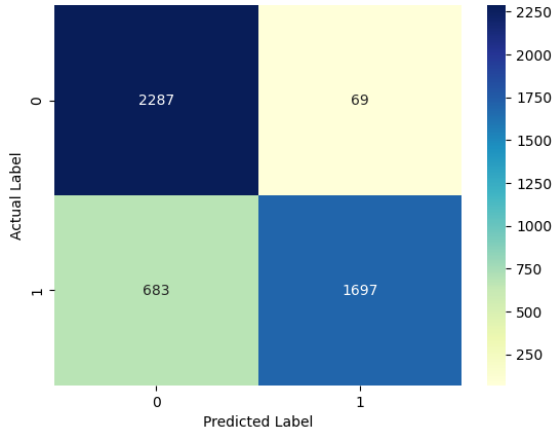


Outliers detected and removed for “Monthly Bill” column

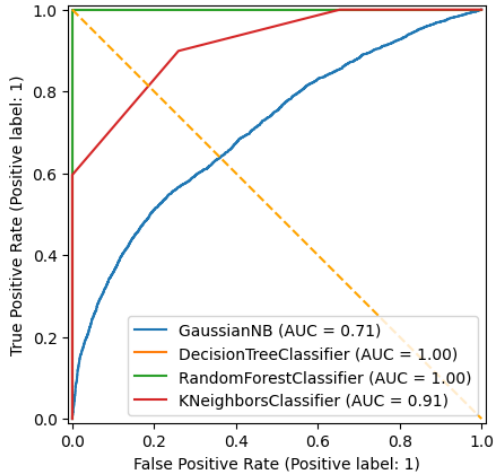
Visualizations for Binary Classification Analysis:



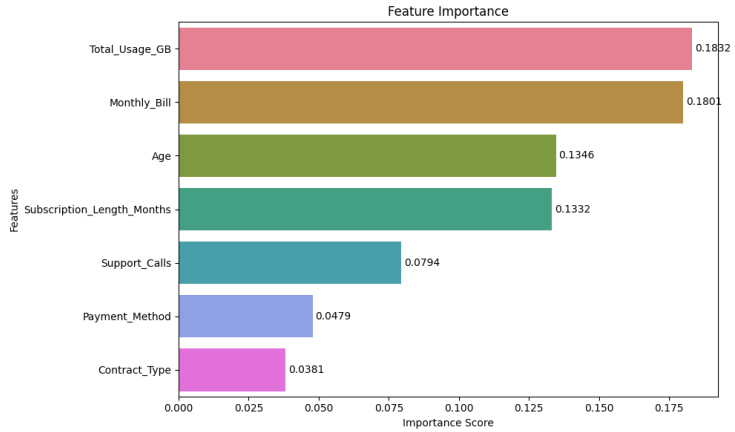
confusion matrix for Naive Bayes



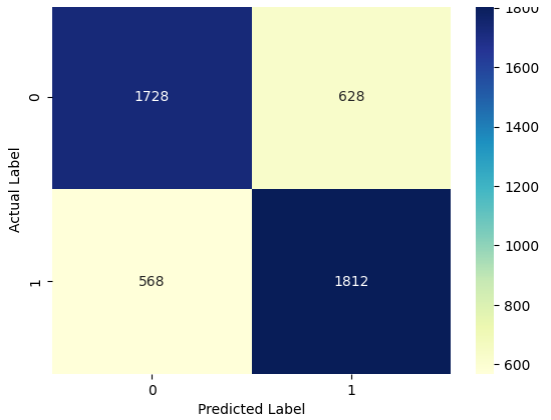
confusion matrix for Random Forest



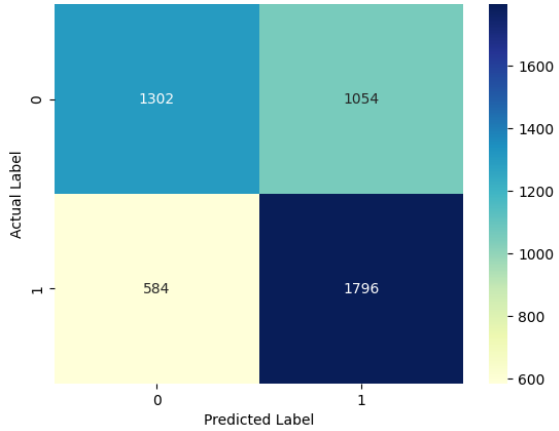
ROC Curve for Training dataset



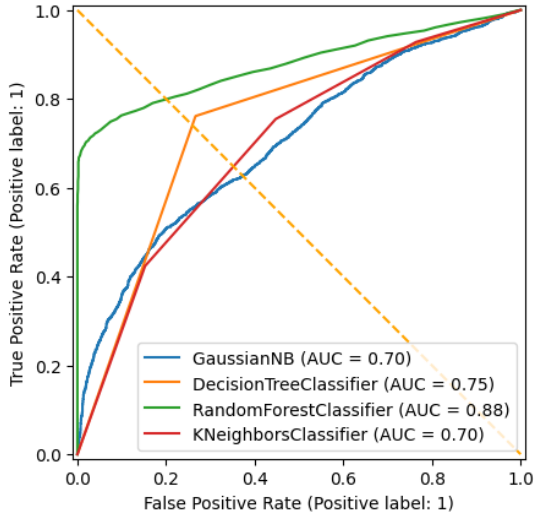
Features importance using Random Forest



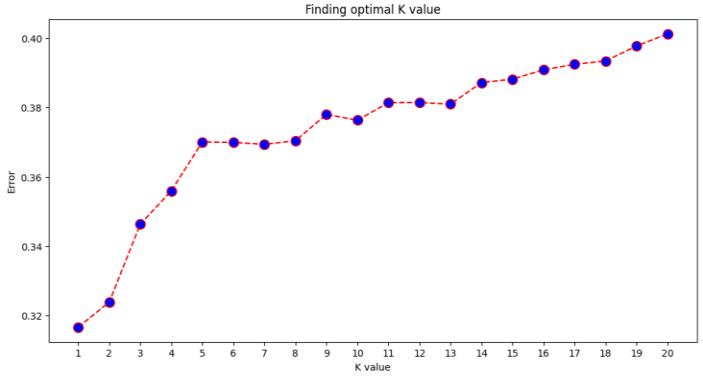
confusion matrix for Decision Tree



confusion matrix for KNN

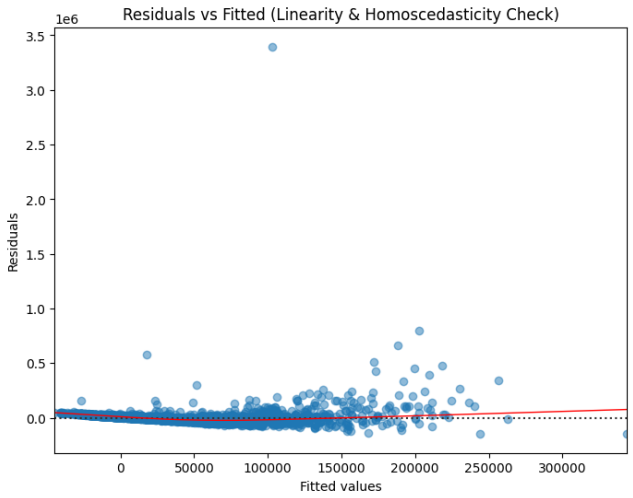


ROC Curve for Testing dataset

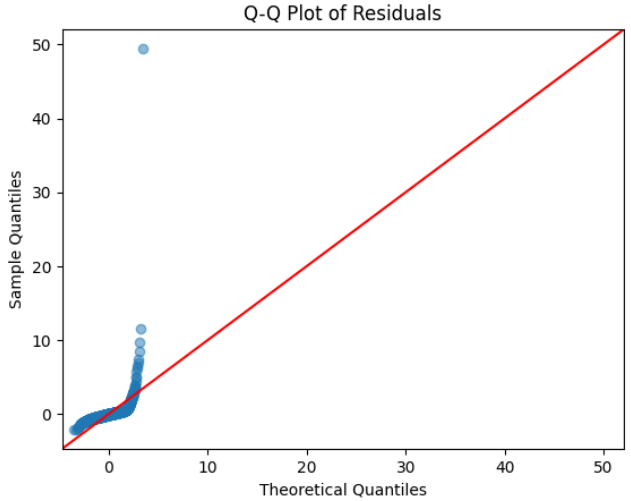


Finding best value of K in KNN

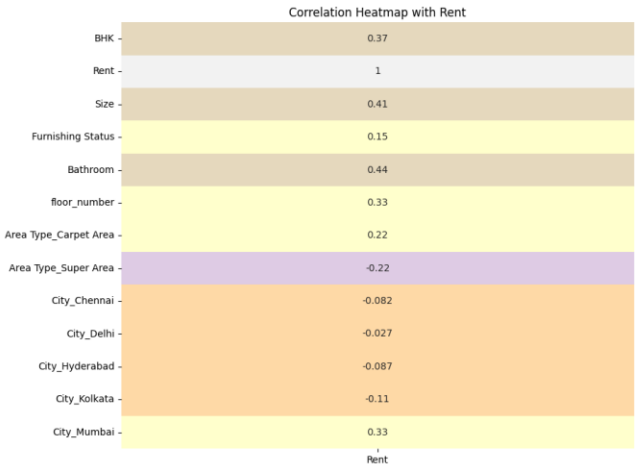
Visualizations for Regression Analysis:



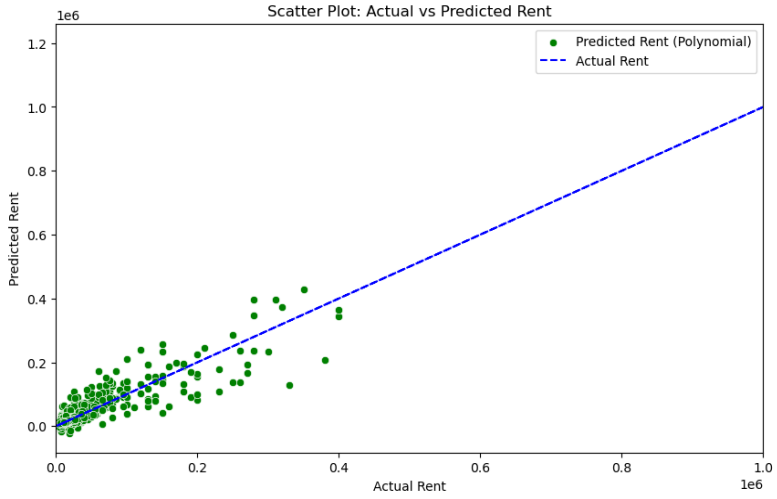
Residuals vs Fitted (Homoscedasticity)



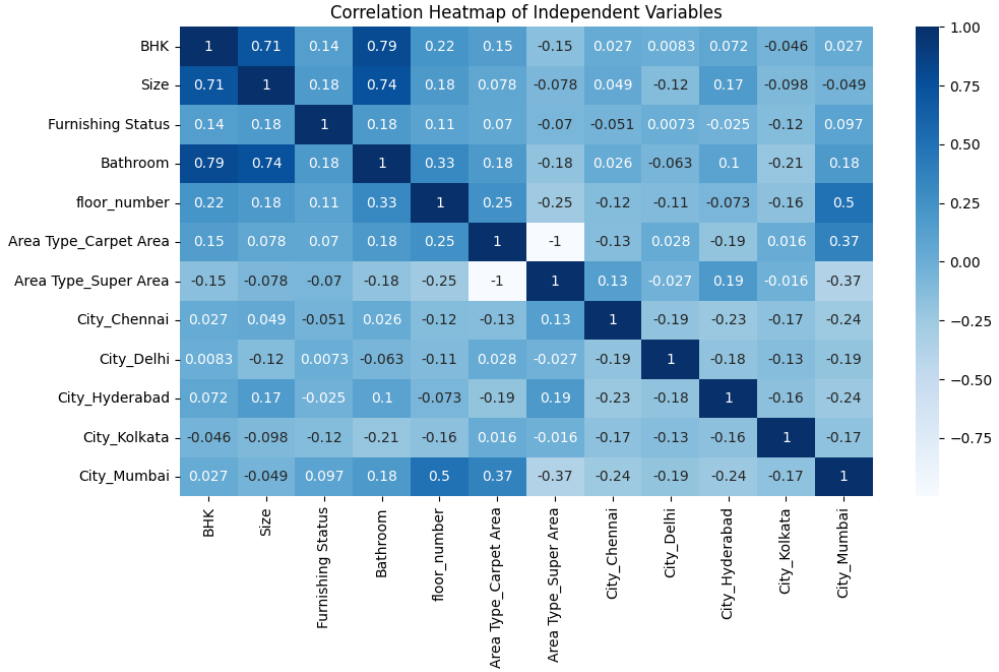
Q-Q Plot of Residuals



Correlation Heatmap with Rent

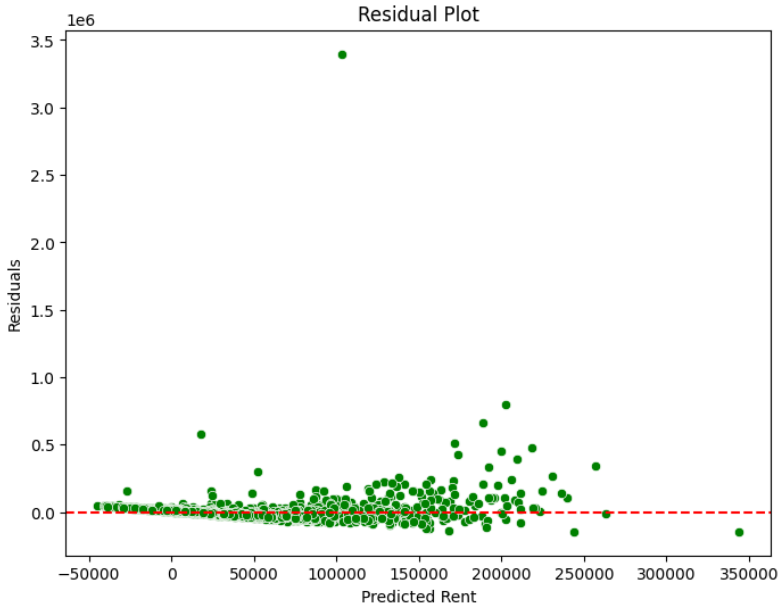


Scatter Plot: Actual vs Predicted Rent

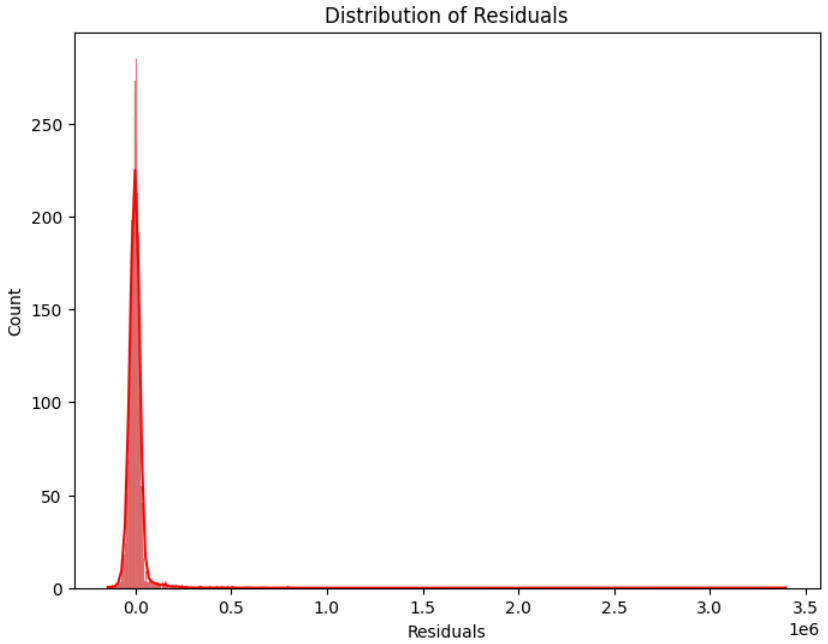


Correlation Heatmap of Independent Variables

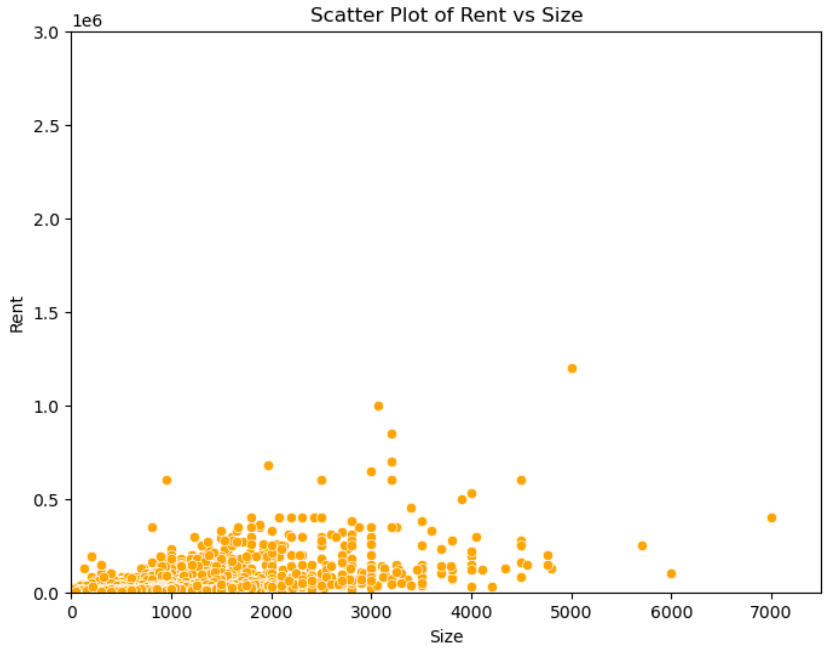
Visualizations for Regression Analysis:



Residual Plot



Distribution of Residuals



Scatter Plot of Rent vs Size

Thank you!