1. Data Preprocessing

In [1]:
```
import pandas as pd
```

1.1 Loading Data & initial data exploration

In [2]:
```
data = pd.read_excel('C:\\Users\\hp\\Downloads\\customer_churn_large_dataset.xlsx')
data
```

Out[2]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	1	Customer_1	63	Male	Los Angeles	17	73.36	236	0
1	2	Customer_2	62	Female	New York	1	48.76	172	0
2	3	Customer_3	24	Female	Los Angeles	5	85.47	460	0
3	4	Customer_4	36	Female	Miami	3	97.94	297	1
4	5	Customer_5	46	Female	Miami	19	58.14	266	0
...	...	...	...	...	...	...	...	...	...
99995	99996	Customer_99996	33	Male	Houston	23	55.13	226	1
99996	99997	Customer_99997	62	Female	New York	19	61.65	351	0
99997	99998	Customer_99998	64	Male	Chicago	17	96.11	251	1
99998	99999	Customer_99999	51	Female	New York	20	49.25	434	1
99999	100000	Customer_100000	27	Female	Los Angeles	19	76.57	173	1

100000 rows × 9 columns

In [3]:
```
data.head(10)
```

Out[3]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	1	Customer_1	63	Male	Los Angeles	17	73.36	236	0
1	2	Customer_2	62	Female	New York	1	48.76	172	0
2	3	Customer_3	24	Female	Los Angeles	5	85.47	460	0
3	4	Customer_4	36	Female	Miami	3	97.94	297	1
4	5	Customer_5	46	Female	Miami	19	58.14	266	0
5	6	Customer_6	67	Male	New York	15	82.65	456	1
6	7	Customer_7	30	Female	Chicago	3	73.79	269	0
7	8	Customer_8	67	Female	Miami	1	97.70	396	1
8	9	Customer_9	20	Female	Miami	10	42.45	150	1
9	10	Customer_10	53	Female	Los Angeles	12	64.49	383	1

In [4]:
```
data.describe()
```

Out[4]:

	CustomerID	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	50000.500000	44.027020	12.490100	65.053197	274.393650	0.497790
std	28867.657797	15.280283	6.926461	20.230696	130.463063	0.499998
min	1.000000	18.000000	1.000000	30.000000	50.000000	0.000000
25%	25000.750000	31.000000	6.000000	47.540000	161.000000	0.000000
50%	50000.500000	44.000000	12.000000	65.010000	274.000000	0.000000
75%	75000.250000	57.000000	19.000000	82.640000	387.000000	1.000000
max	100000.000000	70.000000	24.000000	100.000000	500.000000	1.000000

In [5]:
```
data.dtypes
```

Out[5]:
```
CustomerID                      int64
Name                           object
Age                             int64
Gender                         object
Location                       object
Subscription_Length_Months      int64
Monthly_Bill                  float64
Total_Usage_GB                  int64
Churn                           int64
dtype: object
```

In [6]:
```
data['Churn'].value_counts()
```

Out[6]:
```
0    50221
1    49779
Name: Churn, dtype: int64
```

In [7]:
```
data.nunique()
```

Out[7]:
```
CustomerID                    100000
Name                          100000
Age                               53
Gender                             2
Location                           5
Subscription_Length_Months        24
Monthly_Bill                    7001
Total_Usage_GB                   451
Churn                              2
dtype: int64
```

1.2 Handling missing data

In [8]:
```
data.isna().sum()
```

Out[8]:
```
CustomerID                    0
Name                          0
Age                           0
Gender                        0
Location                      0
Subscription_Length_Months    0
Monthly_Bill                  0
Total_Usage_GB                0
Churn                         0
dtype: int64
```

1.3 Encoding Categorical variable

In [9]:
```
data['Gender'] = data['Gender'].apply(lambda x: 1 if x == 'Male' else 0)
data
```

Out[9]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn
0	1	Customer_1	63	1	Los Angeles	17	73.36	236	0
1	2	Customer_2	62	0	New York	1	48.76	172	0
2	3	Customer_3	24	0	Los Angeles	5	85.47	460	0
3	4	Customer_4	36	0	Miami	3	97.94	297	1
4	5	Customer_5	46	0	Miami	19	58.14	266	0
...	...	...	...	...	...	...	...	...	...
99995	99996	Customer_99996	33	1	Houston	23	55.13	226	1
99996	99997	Customer_99997	62	0	New York	19	61.65	351	0
99997	99998	Customer_99998	64	1	Chicago	17	96.11	251	1
99998	99999	Customer_99999	51	0	New York	20	49.25	434	1
99999	100000	Customer_100000	27	0	Los Angeles	19	76.57	173	1

100000 rows × 9 columns

In [10]:
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
data['LocationNew'] = label_encoder.fit_transform(data['Location'])
data
```

Out[10]:

	CustomerID	Name	Age	Gender	Location	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn	LocationNew
0	1	Customer_1	63	1	Los Angeles	17	73.36	236	0	2
1	2	Customer_2	62	0	New York	1	48.76	172	0	4
2	3	Customer_3	24	0	Los Angeles	5	85.47	460	0	2
3	4	Customer_4	36	0	Miami	3	97.94	297	1	3
4	5	Customer_5	46	0	Miami	19	58.14	266	0	3
...	...	...	...	...	...	...	...	...	...	...
99995	99996	Customer_99996	33	1	Houston	23	55.13	226	1	1
99996	99997	Customer_99997	62	0	New York	19	61.65	351	0	4
99997	99998	Customer_99998	64	1	Chicago	17	96.11	251	1	0
99998	99999	Customer_99999	51	0	New York	20	49.25	434	1	4
99999	100000	Customer_100000	27	0	Los Angeles	19	76.57	173	1	2

100000 rows × 10 columns

In [11]:
```
city_namesE = data['LocationNew'].apply(lambda x: str(x).split(',')[0].strip()).unique()
city_namesE
```

Out[11]:
```
array(['2', '4', '3', '0', '1'], dtype=object)
```

In [12]:
```
city_names = data['Location'].apply(lambda x: str(x).split(',')[0].strip()).unique()
city_names
```

Out[12]:
```
array(['Los Angeles', 'New York', 'Miami', 'Chicago', 'Houston'],
      dtype=object)
```

2. Feature Engineering

In [13]:
```
data.drop(['Name', 'Location'], axis=1, inplace=True)
data
```

Out[13]:

	CustomerID	Age	Gender	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn	LocationNew
0	1	63	1	17	73.36	236	0	2
1	2	62	0	1	48.76	172	0	4
2	3	24	0	5	85.47	460	0	2
3	4	36	0	3	97.94	297	1	3
4	5	46	0	19	58.14	266	0	3
...	...	...	...	...	...	...	...	...
99995	99996	33	1	23	55.13	226	1	1
99996	99997	62	0	19	61.65	351	0	4
99997	99998	64	1	17	96.11	251	1	0
99998	99999	51	0	20	49.25	434	1	4
99999	100000	27	0	19	76.57	173	1	2

100000 rows × 8 columns

In [34]:
```
data['bill_X_GB']= data['Monthly_Bill']*data['Total_Usage_GB']
data['Bill_/_subLen']= data['Monthly_Bill']/data['Subscription_Length_Months']
data['subs_/_bill']= data['Subscription_Length_Months']/data['Monthly_Bill']
data['GB_/_bill']= data['Total_Usage_GB']/data['Monthly_Bill']
data['GB_X_subLen']= data['Total_Usage_GB']*data['Monthly_Bill']
```

Out[34]:

	CustomerID	Age	Gender	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn	LocationNew	bill_X_GB	Bill_/_subLen	subs_/_bill	GB_/_bill	GB_*_bill	GB_X_subLen
0	1	63	1	17	73.36	236	0	2	17312.96	4.315294	0.231734	3.217012	17312.96	17312.96
1	2	62	0	1	48.76	172	0	4	8386.72	48.760000	0.020509	3.527482	8386.72	8386.72
2	3	24	0	5	85.47	460	0	2	39316.20	17.094000	0.058500	5.382005	39316.20	39316.20
3	4	36	0	3	97.94	297	1	3	29088.18	32.646667	0.030631	3.032469	29088.18	29088.18
4	5	46	0	19	58.14	266	0	3	15465.24	3.060000	0.326797	4.575163	15465.24	15465.24
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	99996	33	1	23	55.13	226	1	1	12459.38	2.396957	0.417196	4.099401	12459.38	12459.38
99996	99997	62	0	19	61.65	351	0	4	21639.15	3.244737	0.308191	5.693431	21639.15	21639.15
99997	99998	64	1	17	96.11	251	1	0	24123.61	5.653529	0.176881	2.611591	24123.61	24123.61
99998	99999	51	0	20	49.25	434	1	4	21374.50	2.462500	0.406091	8.812183	21374.50	21374.50
99999	100000	27	0	19	76.57	173	1	2	13246.61	4.030000	0.248139	2.259371	13246.61	13246.61

100000 rows × 14 columns

2.1 Scaling & Normalization

In [35]:
```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
normalized_data = scaler.fit_transform(data)
normalized_df = pd.DataFrame(normalized_data, columns=data.columns)
normalized_df
```

Out[35]:

	CustomerID	Age	Gender	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn	LocationNew	bill_X_GB	Bill_/_subLen	subs_/_bill	GB_/_bill	GB_*_bill	GB_X_subLen
0	0.00000	0.865385	1.0	0.695652	0.619429	0.413333	0.0	0.50	0.326095	0.031043	0.280769	0.168573	0.326095	0.326095
1	0.00001	0.846154	0.0	0.000000	0.268000	0.271111	0.0	1.00	0.141867	0.481209	0.013304	0.187837	0.141867	0.141867
2	0.00002	0.115385	0.0	0.173913	0.792429	0.911111	0.0	0.50	0.780220	0.160475	0.061411	0.302907	0.780220	0.780220
3	0.00003	0.346154	0.0	0.086957	0.970571	0.548889	1.0	0.75	0.569124	0.318002	0.026122	0.157123	0.569124	0.569124
4	0.00004	0.538462	0.0	0.782609	0.402000	0.480000	0.0	0.75	0.287960	0.018329	0.401143	0.252844	0.287960	0.287960
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
99995	0.99996	0.288462	1.0	0.956522	0.359000	0.391111	1.0	0.25	0.225922	0.011613	0.515610	0.223324	0.225922	0.225922
99996	0.99997	0.846154	0.0	0.782609	0.452143	0.668889	0.0	1.00	0.415383	0.020200	0.377583	0.322230	0.415383	0.415383
99997	0.99998	0.884615	1.0	0.695652	0.944429	0.446667	1.0	0.00	0.466660	0.044598	0.211311	0.131008	0.466660	0.466660
99998	0.99999	0.634615	0.0	0.826087	0.275000	0.853333	1.0	1.00	0.409921	0.012277	0.501549	0.515743	0.409921	0.409921
99999	1.00000	0.173077	0.0	0.782609	0.665286	0.273333	1.0	0.50	0.242170	0.028154	0.301542	0.109153	0.242170	0.242170

100000 rows × 14 columns

2.2 Finding Correlation

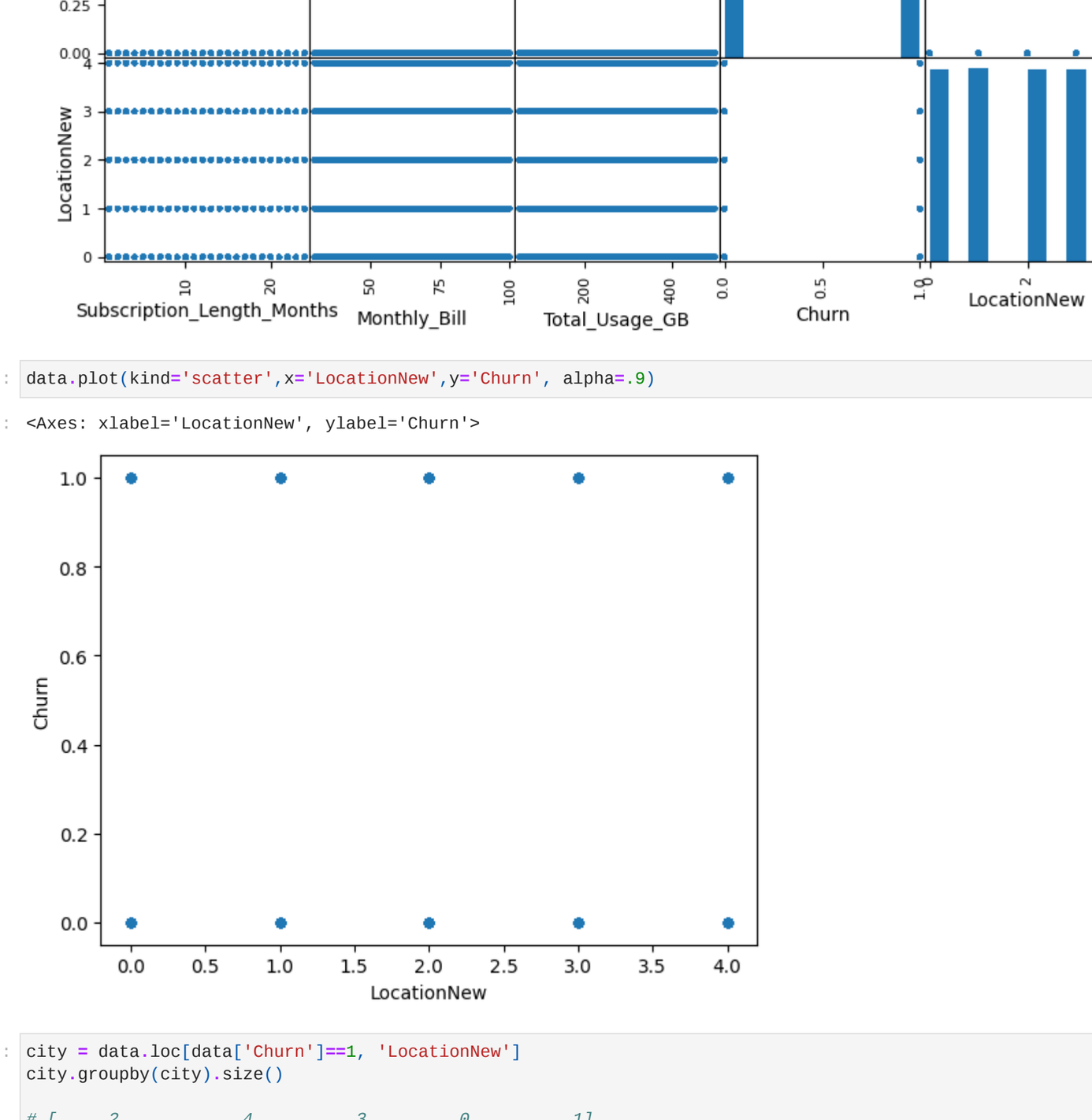
In [36]:
```
corr_matrix = normalized_df.corr()
corr_matrix['Churn'].sort_values(ascending=False)
```

Out[36]:
```
Churn                         1.000000
LocationNew                   0.006405
subs_/_bill                   0.003356
Subscription_Length_Months    0.002328
Gender                        0.002121
Age                           0.001559
Bill_/_subLen                 0.000701
Monthly_Bill                 -0.000211
GB_/_bill                    -0.000718
Total_Usage_GB               -0.002842
bill_X_GB                    -0.003058
GB_*_bill                    -0.003058
GB_X_subLen                  -0.003058
CustomerID                   -0.004586
Name: Churn, dtype: float64
```

In [37]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

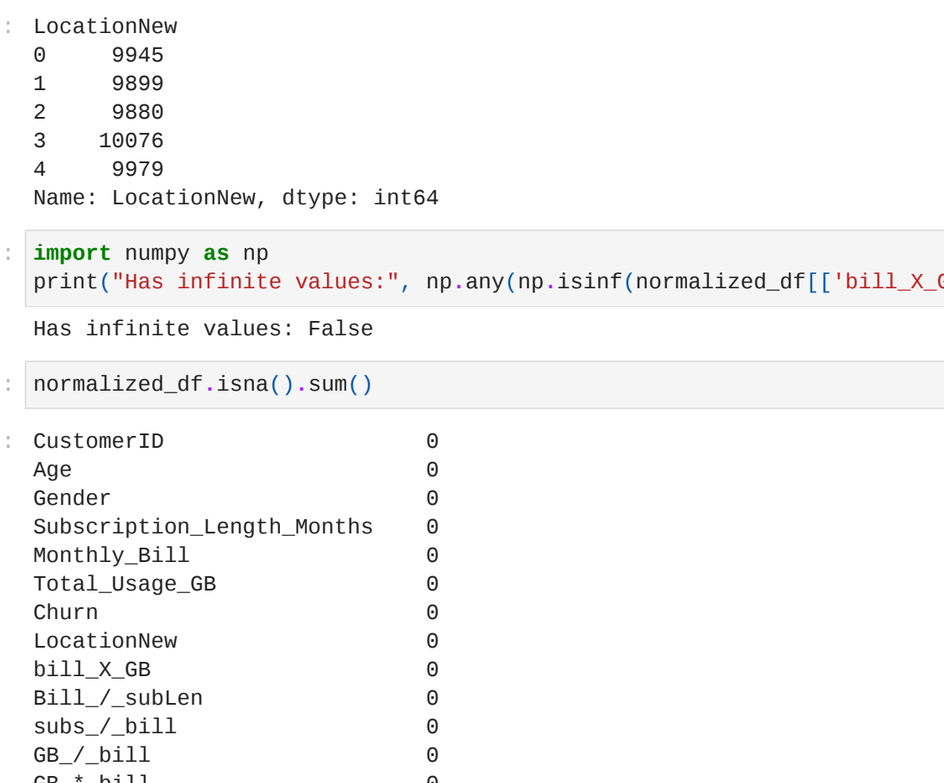
In [38]:
```
from pandas.plotting import scatter_matrix
attributes = ['Subscription_Length_Months','Monthly_Bill','Total_Usage_GB','Churn', 'LocationNew']
scatter_matrix(data[attributes], figsize = (10,10), alpha= 0.3)
```

Out[38]:
```
array([[<Axes: xlabel='Subscription_Length_Months', ylabel='Subscription_Length_Months'>,
        <Axes: xlabel='Monthly_Bill', ylabel='Subscription_Length_Months'>,
        <Axes: xlabel='Total_Usage_GB', ylabel='Subscription_Length_Months'>,
        <Axes: xlabel='Churn', ylabel='Subscription_Length_Months'>,
        <Axes: xlabel='LocationNew', ylabel='Subscription_Length_Months'>],
       [<Axes: xlabel='Subscription_Length_Months', ylabel='Monthly_Bill'>,
        <Axes: xlabel='Monthly_Bill', ylabel='Monthly_Bill'>,
        <Axes: xlabel='Total_Usage_GB', ylabel='Monthly_Bill'>,
        <Axes: xlabel='Churn', ylabel='Monthly_Bill'>,
        <Axes: xlabel='LocationNew', ylabel='Monthly_Bill'>],
       [<Axes: xlabel='Subscription_Length_Months', ylabel='Total_Usage_GB'>,
        <Axes: xlabel='Monthly_Bill', ylabel='Total_Usage_GB'>,
        <Axes: xlabel='Total_Usage_GB', ylabel='Total_Usage_GB'>,
        <Axes: xlabel='Churn', ylabel='Total_Usage_GB'>,
        <Axes: xlabel='LocationNew', ylabel='Total_Usage_GB'>],
       [<Axes: xlabel='Subscription_Length_Months', ylabel='Churn'>,
        <Axes: xlabel='Monthly_Bill', ylabel='Churn'>,
        <Axes: xlabel='Total_Usage_GB', ylabel='Churn'>,
        <Axes: xlabel='Churn', ylabel='Churn'>,
        <Axes: xlabel='LocationNew', ylabel='Churn'>],
       [<Axes: xlabel='Subscription_Length_Months', ylabel='LocationNew'>,
        <Axes: xlabel='Monthly_Bill', ylabel='LocationNew'>,
        <Axes: xlabel='Total_Usage_GB', ylabel='LocationNew'>,
        <Axes: xlabel='Churn', ylabel='LocationNew'>,
        <Axes: xlabel='LocationNew', ylabel='LocationNew'>]], dtype=object)
```



In [39]:
```
data.plot(kind='scatter',x='LocationNew',y='Churn', alpha=.9)
```

Out[39]:
```
<Axes: xlabel='LocationNew', ylabel='Churn'>
```



In [40]:
```
city = data.loc[data['Churn']==1, 'LocationNew']
city.groupby(city).size()

# [     2            4            3            0            1]
# ['Los Angeles', 'New York', 'Miami', 'Chicago', 'Houston']
```

Out[40]:
```
LocationNew
0     9945
1     9899
2     9880
3    10076
4     9979
Name: LocationNew, dtype: int64
```

In [41]:
```
import numpy as np
print("Has infinite values:", np.any(np.isinf(normalized_df[['bill_X_GB','Bill_/_subLen','subs_/_bill']])))
```

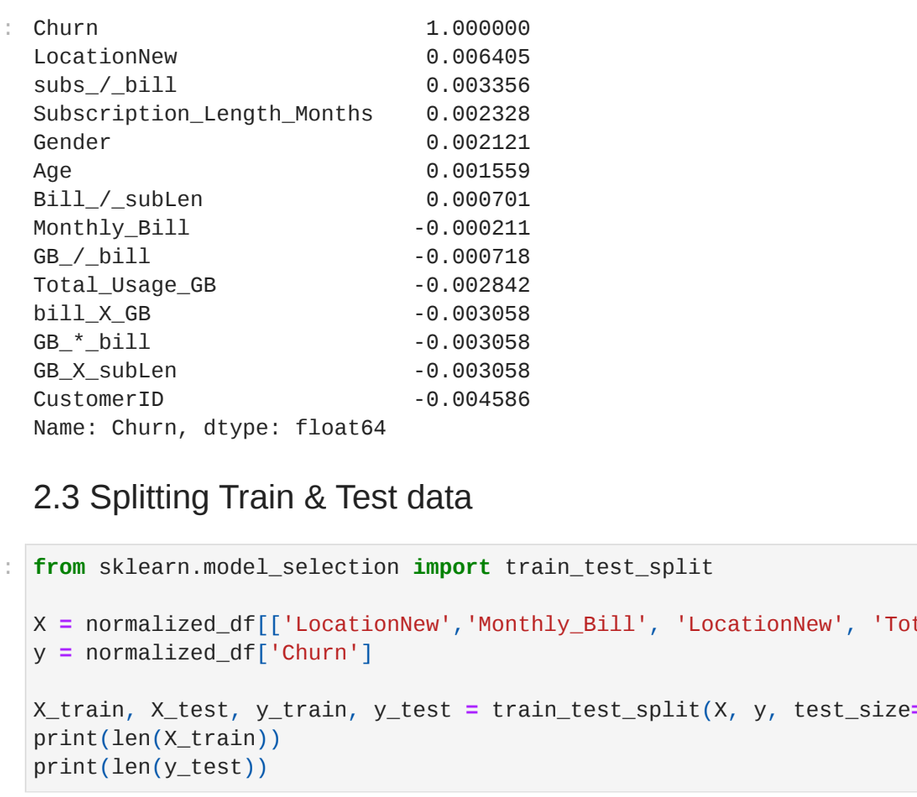
```
Has infinite values: False
```

In [42]:
```
normalized_df.isna().sum()
```

Out[42]:
```
CustomerID                    0
Age                           0
Gender                        0
Subscription_Length_Months    0
Monthly_Bill                  0
Total_Usage_GB                0
Churn                         0
LocationNew                   0
bill_X_GB                     0
Bill_/_subLen                 0
subs_/_bill                   0
GB_/_bill                     0
GB_*_bill                     0
GB_X_subLen                   0
dtype: int64
```

In [43]:
```
normalized_df.plot(kind='scatter',x='subs_/_bill',y='bill_X_GB', alpha=.9)
```

Out[43]:
```
<Axes: xlabel='subs_/_bill', ylabel='bill_X_GB'>
```



In [44]:
```
corr_matrix_2 = normalized_df.corr()
corr_matrix_2['Churn'].sort_values(ascending=False)
```

Out[44]:
```
Churn                         1.000000
LocationNew                   0.006405
subs_/_bill                   0.003356
Subscription_Length_Months    0.002328
Gender                        0.002121
Age                           0.001559
Bill_/_subLen                 0.000701
Monthly_Bill                 -0.000211
GB_/_bill                    -0.000718
Total_Usage_GB               -0.002842
bill_X_GB                    -0.003058
GB_*_bill                    -0.003058
GB_X_subLen                  -0.003058
CustomerID                   -0.004586
Name: Churn, dtype: float64
```

2.3 Splitting Train & Test data

In [45]:
```
from sklearn.model_selection import train_test_split

X = normalized_df[['LocationNew','Monthly_Bill', 'LocationNew', 'Total_Usage_GB', 'bill_X_GB']]
y = normalized_df['Churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print(len(X_train))
print(len(y_test))
```

```
70000
30000
```

In [46]:
```
import numpy as np
print("Has infinite values:", np.any(np.isinf(X)))

max_value = np.max(X)
print("Maximum value in X:", max_value)
```

```
Has infinite values: False
Maximum value in X: LocationNew       1.0
Monthly_Bill      1.0
LocationNew       1.0
Total_Usage_GB    1.0
bill_X_GB         1.0
dtype: float64
```

```
c:\users\hp\appdata\local\programs\python\python38\lib\site-packages\numpy\core\fromnumeric.py:84: FutureWarning: In a future version, DataFrame.max(axis=None) will return a scalar max over the entire DataFrame. To retain the old beh
avior, use 'frame.max(axis=0)' or just 'frame.max()'
  return reduction(axis=axis, out=out, **passkwargs)
```

3 Model Building

3.1 Logistic Regresion

In [47]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Out[47]:
```
0.5011666666666666
```

3.2 Random Forest

In [48]:
```
from sklearn.ensemble import RandomForestClassifier

model2 = RandomForestClassifier(n_estimators= 100, random_state = 42)
model2.fit(X_train, y_train)
model2.score(X_test, y_test)
```

Out[48]:
```
0.5048
```

3.3 Neural Network

In [49]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from sklearn.metrics import accuracy_score


scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)


model = Sequential()

model.add(Dense(units=64, activation='relu', input_dim=X_scaled.shape[1]))

model.add(Dense(units=32, activation='relu'))

model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1, verbose=1)

y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)


accuracy = accuracy_score(y_test, y_pred_classes)
print("Accuracy:", accuracy)
```

```
Epoch 1/10
2250/2250 [==============================] - 11s 4ms/step - loss: 0.6951 - accuracy: 0.4978 - val_loss: 0.6940 - val_accuracy: 0.5001
Epoch 2/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6937 - accuracy: 0.5019 - val_loss: 0.6933 - val_accuracy: 0.5015
Epoch 3/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6934 - accuracy: 0.5021 - val_loss: 0.6935 - val_accuracy: 0.4916
Epoch 4/10
2250/2250 [==============================] - 9s 4ms/step - loss: 0.6932 - accuracy: 0.4996 - val_loss: 0.6933 - val_accuracy: 0.4938
Epoch 5/10
2250/2250 [==============================] - 10s 5ms/step - loss: 0.6932 - accuracy: 0.5043 - val_loss: 0.6932 - val_accuracy: 0.4965
Epoch 6/10
2250/2250 [==============================] - 10s 5ms/step - loss: 0.6932 - accuracy: 0.5003 - val_loss: 0.6932 - val_accuracy: 0.5035
Epoch 7/10
2250/2250 [==============================] - 10s 5ms/step - loss: 0.6932 - accuracy: 0.5041 - val_loss: 0.6932 - val_accuracy: 0.4997
Epoch 8/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5027 - val_loss: 0.6935 - val_accuracy: 0.5013
Epoch 9/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6932 - accuracy: 0.5046 - val_loss: 0.6932 - val_accuracy: 0.4984
Epoch 10/10
2250/2250 [==============================] - 10s 5ms/step - loss: 0.6931 - accuracy: 0.5036 - val_loss: 0.6934 - val_accuracy: 0.4988
625/625 [==============================] - 2s 3ms/step
Accuracy: 0.50275
```

3.4 Model Performance

In [50]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report

model.fit(X_train, y_train, epochs=10, batch_size=32, validation_split=0.1, verbose=1)

y_pred = model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred_classes)
precision = precision_score(y_test, y_pred_classes)
recall = recall_score(y_test, y_pred_classes)
f1 = f1_score(y_test, y_pred_classes)
confusion = confusion_matrix(y_test, y_pred_classes)
classification_rep = classification_report(y_test, y_pred_classes)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("Confusion Matrix:\n", confusion)
print("Classification Report:\n", classification_rep)
```

```
Epoch 1/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5038 - val_loss: 0.6933 - val_accuracy: 0.4991
Epoch 2/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5050 - val_loss: 0.6932 - val_accuracy: 0.4996
Epoch 3/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5033 - val_loss: 0.6932 - val_accuracy: 0.4995
Epoch 4/10
2250/2250 [==============================] - 10s 5ms/step - loss: 0.6931 - accuracy: 0.5051 - val_loss: 0.6937 - val_accuracy: 0.4984
Epoch 5/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6932 - accuracy: 0.5038 - val_loss: 0.6932 - val_accuracy: 0.4997
Epoch 6/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5050 - val_loss: 0.6932 - val_accuracy: 0.4990
Epoch 7/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5047 - val_loss: 0.6933 - val_accuracy: 0.4994
Epoch 8/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5052 - val_loss: 0.6933 - val_accuracy: 0.4983
Epoch 9/10
2250/2250 [==============================] - 11s 5ms/step - loss: 0.6931 - accuracy: 0.5038 - val_loss: 0.6933 - val_accuracy: 0.4992
Epoch 10/10
2250/2250 [==============================] - 10s 4ms/step - loss: 0.6931 - accuracy: 0.5048 - val_loss: 0.6933 - val_accuracy: 0.5027
625/625 [==============================] - 1s 2ms/step
Accuracy: 0.50335
Precision: 0.49843260188087773
Recall: 0.1923193226489265
F1-score: 0.2775474579969452
Confusion Matrix:
 [[8159 1920]
 [8013 1908]]
Classification Report:
               precision    recall  f1-score   support

         0.0       0.50      0.81      0.62     10079
         1.0       0.50      0.19      0.28      9921

    accuracy                           0.50     20000
   macro avg       0.50      0.50      0.45     20000
weighted avg       0.50      0.50      0.45     20000
```

4. Model Optimization

4.1 Fine Tuning

In [51]:
```
from keras.models import load_model
from keras.layers import Dense
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import accuracy_score



model = Sequential()

model.add(Dense(units=128, activation='relu', input_dim=X_scaled.shape[1]))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=1, activation='sigmoid'))

model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])

early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True)

history = model.fit(
    X_train, y_train,
    epochs=10,
    batch_size=64,
    validation_split=0.1,
    callbacks=[early_stopping, model_checkpoint],
    verbose=1
)

best_model = load_model('best_model.h5')

y_pred = best_model.predict(X_test)
y_pred_classes = (y_pred > 0.5).astype(int)

accuracy = accuracy_score(y_test, y_pred_classes)
print("Accuracy:", accuracy)
```

```
Epoch 1/10
1125/1125 [==============================] - 8s 6ms/step - loss: 0.6944 - accuracy: 0.5017 - val_loss: 0.6945 - val_accuracy: 0.5000
Epoch 2/10
1125/1125 [==============================] - 5s 4ms/step - loss: 0.6936 - accuracy: 0.5004 - val_loss: 0.6933 - val_accuracy: 0.5098
Epoch 3/10
1125/1125 [==============================] - 5s 5ms/step - loss: 0.6934 - accuracy: 0.5001 - val_loss: 0.6935 - val_accuracy: 0.5016
Epoch 4/10
1125/1125 [==============================] - 6s 5ms/step - loss: 0.6933 - accuracy: 0.5025 - val_loss: 0.6940 - val_accuracy: 0.4949
Epoch 5/10
1125/1125 [==============================] - 6s 5ms/step - loss: 0.6932 - accuracy: 0.5022 - val_loss: 0.6933 - val_accuracy: 0.5010
Epoch 6/10
1125/1125 [==============================] - 6s 6ms/step - loss: 0.6932 - accuracy: 0.5024 - val_loss: 0.6933 - val_accuracy: 0.5040
Epoch 7/10
1125/1125 [==============================] - 5s 5ms/step - loss: 0.6932 - accuracy: 0.5023 - val_loss: 0.6933 - val_accuracy: 0.5059
Epoch 8/10
1125/1125 [==============================] - 6s 5ms/step - loss: 0.6932 - accuracy: 0.5037 - val_loss: 0.6932 - val_accuracy: 0.5017
Epoch 9/10
1125/1125 [==============================] - 6s 5ms/step - loss: 0.6931 - accuracy: 0.5036 - val_loss: 0.6933 - val_accuracy: 0.4966
Epoch 10/10
1125/1125 [==============================] - 5s 5ms/step - loss: 0.6932 - accuracy: 0.4994 - val_loss: 0.6933 - val_accuracy: 0.4997
625/625 [==============================] - 2s 3ms/step
Accuracy: 0.50235
```

4.3 Hyperparameter Tuning

In [ ]:
```
from sklearn.model_selection import StratifiedKFold, cross_val_score, GridSearchCV
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.wrappers.scikit_learn import KerasClassifier


# Function to create the ANN model
def create_ann(learning_rate=0.001):
    model = Sequential()
    model.add(Dense(units=128, activation='relu', input_dim=X_scaled.shape[1]))
    model.add(Dense(units=64, activation='relu'))
    model.add(Dense(units=1, activation='sigmoid'))
    model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])
    return model

# Create a KerasClassifier with adjustable parameters
keras_classifier = KerasClassifier(build_fn=create_ann, verbose=0)

# Perform stratified k-fold cross-validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cross_val_results = cross_val_score(keras_classifier, X_scaled, y, cv=cv, scoring='accuracy')

print("Cross-validation results:", cross_val_results)
print("Mean Accuracy:", cross_val_results.mean())
print("Standard Deviation:", cross_val_results.std())

# Define hyperparameters and their possible values
param_grid = {
    'epochs': [10, 13],
    'batch_size': [32, 64],
    'learning_rate': [0.001, 0.01]
}

# Perform grid search
grid = GridSearchCV(estimator=keras_classifier, param_grid=param_grid, scoring='accuracy', cv=cv)
grid_result = grid.fit(X_scaled, y)

print("Best Score:", grid_result.best_score_)
print("Best Params:", grid_result.best_params_)
```

```
C:\Users\hp\AppData\Local\Temp\ipykernel_21080\1165629039.py:18: DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras (https://github.com/adriangb/scikeras) instead. See https://www.adriangb.com/scikeras/stable/migration.
html for help migrating.
  keras_classifier = KerasClassifier(build_fn=create_ann, verbose=0)
```

```
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 1s 2ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
Cross-validation results: [0.5013  0.50545 0.4987  0.5047  0.50095]
Mean Accuracy: 0.50222
Standard Deviation: 0.002507309314783432
625/625 [==============================] - 1s 2ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
625/625 [==============================] - 2s 3ms/step
```