In [42]:

```python
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
from sklearn.metrics import accuracy_score
from sklearn.ensemble import RandomForestRegressor
```

In [43]:

```python
df = pd.read_csv("vegiedata.csv")
df.head(10)
```

Out[43]:

	Sl no.	District Name	Market Name	Commodity	Variety	Grade	Min Price	Max Price	Modal Price	Date
0	1	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	30-Dec-22
1	2	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	29-Dec-22
2	3	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	28-Dec-22
3	4	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	23-Dec-22
4	5	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	22-Dec-22
5	6	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	21-Dec-22
6	7	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	20-Dec-22
7	8	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	19-Dec-22
8	9	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	17-Dec-22
9	10	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	15-Dec-22

In [44]:

```python
import pandas as pd


# Create a dictionary to map the old column names to the new ones
column_mapping = {'District Name': 'district', 'Market Name': 'market', 'Date': 'arrival_date',
                  'Max Price': 'max_price','Min Price': 'min_price','Modal Price': 'modal_price',
                  'Commodity': 'commodity','Variety': 'variety'}

# Rename the columns using the mapping dictionary
df.rename(columns=column_mapping, inplace=True)
```

In [45]:

```python
data = df
```

In [ ]:

In [ ]:

In [46]:

```python
data
```

Out[46]:

	Sl no.	district	market	commodity	variety	Grade	min_price	max_price	modal_price	arrival_date
0	1	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	30-Dec-22
1	2	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	29-Dec-22
2	3	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	28-Dec-22
3	4	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	23-Dec-22
4	5	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	22-Dec-22
...	...	...	...	...	...	...	...	...	...	...
5180	5183	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5181	5184	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5182	5185	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5183	5186	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5184	5187	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5185 rows × 10 columns

In [47]:

```python
data = data.dropna()
```

In [48]:

```python
data
```

Out[48]:

	Sl no.	district	market	commodity	variety	Grade	min_price	max_price	modal_price	arrival_date
0	1	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	30-Dec-22
1	2	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	29-Dec-22
2	3	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	28-Dec-22
3	4	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	23-Dec-22
4	5	Adilabad	Adilabad(Rythu Bazar)	Tomato	Other	FAQ	1500.0	1500.0	1500.0	22-Dec-22
...	...	...	...	...	...	...	...	...	...	...
3993	3996	Mahbubnagar	Kalwakurthy	Brinjal	Brinjal	FAQ	500.0	1500.0	1000.0	20-May-22
3994	3997	Mahbubnagar	Kalwakurthy	Brinjal	Brinjal	FAQ	500.0	1100.0	800.0	19-May-22
3995	3998	Mahbubnagar	Kalwakurthy	Brinjal	Brinjal	FAQ	600.0	1500.0	1050.0	18-May-22
3996	3999	Mahbubnagar	Kalwakurthy	Brinjal	Brinjal	FAQ	1000.0	1500.0	1250.0	17-May-22
3997	4000	Mahbubnagar	Kalwakurthy	Brinjal	Brinjal	FAQ	1000.0	1500.0	1250.0	16-May-22

3998 rows × 10 columns

In [49]:

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3998 entries, 0 to 3997
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   Sl no.        3998 non-null   int64  
 1   district      3998 non-null   object 
 2   market        3998 non-null   object 
 3   commodity     3998 non-null   object 
 4   variety       3998 non-null   object 
 5   Grade         3998 non-null   object 
 6   min_price     3998 non-null   float64
 7   max_price     3998 non-null   float64
 8   modal_price   3998 non-null   float64
 9   arrival_date  3998 non-null   object 
dtypes: float64(3), int64(1), object(6)
memory usage: 343.6+ KB
```

In [50]:

```python
data['min_price'] = data['min_price']/100
data['max_price'] = data['max_price']/100
data['modal_price'] = data['modal_price'] / 100
```

```
C:\Users\koppi_kutdpvm\AppData\Local\Temp\ipykernel_26604\3474568116.py:1: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['min_price'] = data['min_price']/100
C:\Users\koppi_kutdpvm\AppData\Local\Temp\ipykernel_26604\3474568116.py:2: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['max_price'] = data['max_price']/100
C:\Users\koppi_kutdpvm\AppData\Local\Temp\ipykernel_26604\3474568116.py:3: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data['modal_price'] = data['modal_price'] / 100
```

In [51]:

```python
X1 = data.iloc[:,:8]
```

In [52]:

```python
Y1 = data['modal_price']
```

In [53]:

```python
X = data[['commodity','min_price','max_price']]
Y = data['modal_price']
cat_mask = (X.dtypes==object)
cat_cols = X.columns[cat_mask].tolist()
le = LabelEncoder()
X[cat_cols] = X[cat_cols].apply(lambda x:le.fit_transform(x))
```

```
C:\Users\koppi_kutdpvm\AppData\Local\Temp\ipykernel_26604\1597142809.py:6: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[cat_cols] = X[cat_cols].apply(lambda x:le.fit_transform(x))
```

In [54]:

```python
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=42)
```

In [55]:

```python
def xgb_model():
    st = StandardScaler()
    xgb_reg = xgb.XGBRegressor()
    steps = [('scaler',st),('model',xgb_reg)]
    xgb_pipeline = Pipeline(steps)

    param = {
        'model__subsample' : np.arange(0.05,1.05),
        'model__max_depth': np.arange(3,20,1),
        'model__colsample_bytree':np.arange(.1,1.05,.05),
        'model__learning_rate': np.arange(0,1,.1)
    }
    rand = RandomizedSearchCV(estimator=xgb_pipeline,param_distributions = param,n_iter=3,scoring='neg_mean_squared_error',cv=4)
    rand.fit(X_train,Y_train)
    model = rand.best_estimator_
    return model
```

In [56]:

```python
model = xgb_model()
```

In [57]:

```python
model.fit(X_train,Y_train)
```

Out[57]:

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('model',
                 XGBRegressor(base_score=None, booster=None, callbacks=None,
                              colsample_bylevel=None, colsample_bynode=None,
                              colsample_bytree=0.3500000000000001,
                              early_stopping_rounds=None,
                              enable_categorical=False, eval_metric=None,
                              feature_types=None, gamma=None, gpu_id=None,
                              grow_policy=None, importance_type=None,
                              interaction_constraints=None, learning_rate=0.4,
                              max_bin=None, max_cat_threshold=None,
                              max_cat_to_onehot=None, max_delta_step=None,
                              max_depth=14, max_leaves=None,
                              min_child_weight=None, missing=nan,
                              monotone_constraints=None, n_estimators=100,
                              n_jobs=None, num_parallel_tree=None,
                              predictor=None, random_state=None, ...))])
```

In [58]:

```python
model.score(X_test,Y_test)
```

Out[58]:

```
0.985348328371898
```

In [59]:

```python
model1 = xgb_model()
```

In [60]:

```python
model1.fit(X_train,Y_train)
```

Out[60]:

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('model',
                 XGBRegressor(base_score=None, booster=None, callbacks=None,
                              colsample_bylevel=None, colsample_bynode=None,
                              colsample_bytree=0.9000000000000002,
                              early_stopping_rounds=None,
                              enable_categorical=False, eval_metric=None,
                              feature_types=None, gamma=None, gpu_id=None,
                              grow_policy=None, importance_type=None,
                              interaction_constraints=None, learning_rate=0.1,
                              max_bin=None, max_cat_threshold=None,
                              max_cat_to_onehot=None, max_delta_step=None,
                              max_depth=8, max_leaves=None,
                              min_child_weight=None, missing=nan,
                              monotone_constraints=None, n_estimators=100,
                              n_jobs=None, num_parallel_tree=None,
                              predictor=None, random_state=None, ...))])
```

In [79]:

```python
model1.score(X_test,Y_test)
```

Out[79]:

```
0.9885496845820512
```

In [98]:

```python
import pickle
import xgboost as xgb



# Save the trained model to a file
#filename = 'xgboost_model.pkl'
#pickle.dump(model, open(filename, 'wb'))

# Load the saved model from the file
#loaded_model = pickle.load(open(filename, 'rb'))
```

In [86]:

```python
import pickle
import xgboost as xgb

# Save the trained model to a file
#filename = 'xgb_model1.sav'
#pickle.dump(model1, open(filename, 'wb'))
```

In [133…

```python
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.tree import DecisionTreeRegressor

# Split the data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Encode categorical features
cat_mask = (X_train.dtypes == object)
cat_cols = X_train.columns[cat_mask].tolist()
le = LabelEncoder()
X_train[cat_cols] = X_train[cat_cols].apply(lambda x: le.fit_transform(x))

# Scale the numerical features
st = StandardScaler()
X_train[['min_price', 'max_price']] = st.fit_transform(X_train[['min_price', 'max_price']])

# Define the XGBoost regressor
model = DecisionTreeRegressor()

# Train the XGBoost regressor
model.fit(X_train, Y_train)

# Encode categorical features in the test data
X_test[cat_cols] = X_test[cat_cols].apply(lambda x: le.transform(x))

# Scale the numerical features in the test data
X_test[['min_price', 'max_price']] = st.transform(X_test[['min_price', 'max_price']])

# Make predictions using the trained model
predictions = model.predict(X_test)

# Display the predictions
print("Predicted modal prices:", predictions)
```

```
Predicted modal prices: [28.         44.         40.          7.95238095 50.07142857  7.6
  7.26086957  7.60294118 10.42105263 30.         40.          7.87037037
 36.         12.1402439  34.          7.26086957 12.          5.75
 20.         18.          3.         40.         35.         26.25
 20.          3.5        40.16666667 30.          3.          5.86477273
 35.          5.86477273 10.         30.          7.60294118  7.26086957
 17.5        13.          5.86477273 12.         15.31147541 22.
  7.26086957 11.          4.18181818 10.42105263 14.         20.
  3.          9.         17.78333333 35.         20.          7.91666667
  8.89473684 13.25       50.07142857 44.          7.26086957  4.65
  6.05405405  5.         20.         44.          7.95238095 15.01408451
 25.02469136 15.         18.5        18.          5.86477273 23.
 10.         22.         13.         22.8        12.         36.
  5.75       40.         50.07142857 15.01408451 50.07142857 25.02469136
 14.66666667  8.67741935  5.97058824  4.83333333 11.         50.07142857
  3.82352941 15.3        44.         27.06666667 40.         15.01408451
 70.         24.         18.          5.60869565 27.06666667 18.
  6.15       13.75       30.         15.33333333  5.60869565 11.5
  7.26086957  5.         23.         45.         15.375      15.01408451
  7.26086957 24.         16.         40.         10.625      12.
 20.         40.         12.1402439  45.         26.5         5.
 30.         11.         10.         10.625      35.          7.26086957
 15.04347826  3.33333333 15.04347826 50.07142857 12.22222222 21.
  7.95238095 22.75       15.01408451 10.          7.26086957 15.01408451
 18.4        10.         20.         10.         20.         45.
 12.1402439  15.92307692 22.8        15.33333333  7.95238095 10.75862069
 48.          6.05405405 32.         15.31147541 21.         40.
 25.02469136 13.         22.23333333 28.         42.5         6.05405405
  7.26086957 40.16666667 44.          2.          7.91666667 30.
  3.         19.         27.9        15.01408451 30.         50.07142857
 60.         50.         11.25        3.         23.         10.
  8.5        18.         15.31147541 15.01408451 15.25        7.91666667
 25.02469136  8.67741935 10.42105263  7.95238095 28.         24.5
  5.86477273 15.31147541 58.         20.         40.16666667  5.86477273
 30.          7.60294118 15.01408451 15.33333333  6.6        15.31147541
 17.78333333 22.8        38.         22.23529412 70.          8.5
 18.          7.26086957 44.         15.31147541  4.83333333 17.7972973
 40.         10.42105263  3.5        50.          5.86477273 12.1402439
 70.         15.54545455 35.         16.         64.         15.31147541
  6.75       25.         20.         14.66666667 35.         15.31147541
 31.66666667 10.         14.8         4.         52.          5.86477273
 44.         60.         15.31147541 15.04347826  7.26086957  7.
 17.7972973  15.04347826 12.1402439  20.          6.6        15.54545455
 13.         13.75       11.         11.31944444 15.31147541 31.25
 45.          7.95238095  5.86477273 17.         22.23529412 15.01408451
 17.7972973  13.         11.31944444 11.5        15.33333333 35.
 17.          2.25       50.         11.5         6.25        7.26086957
  5.86477273  6.05405405 15.         55.         15.         25.02469136
 22.23333333 26.27777778 44.         45.         24.5        23.
 50.         60.         10.75862069 40.         20.         45.
 45.          7.91666667 20.          7.91666667 14.66666667 10.
 15.31147541 10.66666667 48.         16.         17.7972973  16.33333333
 40.          3.         17.78333333  5.86477273 22.         30.
 10.625       5.          4.         14.         17.7972973   4.65
 25.02469136  7.26086957 10.         12.1402439  70.         25.02469136
  7.91666667 20.         15.31147541  7.91666667 22.          8.89473684
 30.09090909 44.         40.         15.33333333 15.01408451 30.
 21.25       22.23529412 15.31147541  5.8         5.86477273 12.1402439
 28.         80.          7.91666667 30.         14.5        10.42105263
 13.         15.33333333 50.07142857  7.60294118 50.         31.91666667
 27.9        19.33333333  4.         20.         10.5        31.66666667
  3.         18.5        12.         10.         31.91666667 40.
  7.91666667  3.         40.16666667 13.         35.         20.
 15.33333333  7.9        20.          3.         10.         14.
  7.95238095 40.          6.          2.5        23.          3.
 25.02469136 17.7972973  15.31147541 35.         12.1402439   7.95238095
 28.         20.         10.         35.         41.          7.91666667
 70.         15.01408451 18.42857143 16.          7.95238095 35.
 12.22222222  6.875       7.60294118 18.         12.         27.4
  4.65       50.07142857 24.5        15.92307692 13.75        8.
  3.         40.16666667 40.16666667 22.75       15.54545455 44.
 31.66666667 25.02469136 50.          7.26086957  5.86477273 40.
 28.         40.          6.05405405 50.          6.05405405 40.
 50.07142857 25.02469136 28.         12.1402439  30.          5.60869565
  5.60869565 10.         40.16666667  5.97058824  7.26086957 15.01408451
 15.92307692 16.         30.         21.66666667 22.8        12.
 14.66666667 10.         11.         15.01408451 23.14285714  8.67741935
 22.23529412  7.95238095 15.          2.5        10.75862069 25.02469136
 10.5        27.06666667  1.         10.         50.07142857 40.
 22.75        5.60869565  8.67741935 11.31944444  8.89473684 12.
  3.         25.          5.60869565 11.31944444  3.33333333 10.
 13.25        7.26086957 50.         15.33333333  7.60294118 18.2
  3.5        32.         15.33333333  7.91666667  6.05405405  5.60869565
 11.31944444 22.         15.33333333  7.91666667 10.         13.
 42.5        5.97058824 14.66666667 22.75       25.17857143  8.89473684
 15.31147541 22.23529412 32.          8.75        6.05405405  7.26086957
  3.          6.05405405 18.         15.31147541 22.8        17.7972973
 50.07142857 25.         12.1402439  15.31147541 28.         27.33333333
  8.67741935 22.23333333 23.          7.9         7.26086957 10.5
  8.67741935 31.25        7.26086957  9.75        4.         21.25
 11.25       10.42105263 40.         22.23529412  5.97058824 10.42105263
 20.         15.01408451  7.60294118 10.625      13.         20.
 22.8        20.          7.91666667  2.5        36.         15.
  5.8         4.18181818 10.         25.02469136 44.          6.05405405
 13.          9.          5.86477273 18.42857143 15.33333333 35.
 10.42105263  2.5        12.1402439   7.26086957 20.         15.01408451
 18.          7.95238095  3.82352941 12.1402439   8.         30.
 17.78333333 10.625      20.          9.         15.54545455 25.02469136
 20.5        5.          9.9047619  32.          5.86477273  7.87037037
 70.         15.31147541  7.26086957 10.42105263  4.18181818 14.66666667
 45.         25.02469136 15.         20.         15.31147541 17.7972973
 50.07142857 15.          6.05405405 35.         13.          7.91666667
 18.         44.         25.02469136 10.66666667 40.16666667 10.
 11.31944444 10.         15.3        20.          6.05405405  1.
 12.1402439  18.         15.31147541 21.66666667  7.95238095  5.86477273
 15.01408451  7.91666667  7.26086957 26.66666667 39.         25.02469136
  6.05405405  7.95238095  5.86477273 52.         22.          7.91666667
 25.02469136 12.1402439  27.06666667 27.66666667 18.2         6.15
 10.66666667 15.01408451 30.         35.          2.25       25.02469136
  2.5        27.06666667 13.66666667 44.         27.88888889  8.5
 12.         26.27777778 24.2        70.         10.         17.78333333
 60.          2.         27.33333333 22.          8.89473684 58.
  8.89473684  5.         23.14285714 30.         22.         10.625
  9.9047619  23.         16.33333333  5.86477273 11.31944444 44.
 10.          4.18181818 33.75       25.         11.31944444 32.5
  5.86477273 25.02469136 32.         22.8        14.8        27.66666667
 15.01408451 20.         14.8        48.         12.1402439  17.78333333
  5.86477273 28.         24.          7.91666667 15.31147541 17.78333333
 29.          5.97058824 15.01408451 25.02469136 22.8        25.
 31.66666667 15.01408451  4.18181818  7.91666667 35.          7.95238095
 10.         19.33333333 11.5        30.         30.09090909 35.
  7.26086957 10.          9.2        15.01408451 20.         30.
 44.         13.         15.31147541 27.06666667  9.5        10.75862069
  7.91666667  4.          2.5         4.83333333 26.27777778 90.
  7.95238095 26.66666667 27.33333333  6.05405405  8.67741935 12.1402439
 44.         10.42105263 25.02469136 18.         16.25       30.
 11.5        25.02469136  6.05405405 13.5        80.         14.66666667
  7.95238095 20.         25.17857143 55.          7.26086957 15.
 14.25       23.          7.91666667 12.22222222  7.26086957 40.
 10.         40.         25.02469136  6.05405405  5.86477273 31.25
 48.         18.42857143  5.86477273 13.25       48.         27.88888889
  7.26086957  4.18181818 18.          6.05405405 11.31944444 36.
 30.09090909 32.5         4.65       12.         15.          7.
  7.87037037  9.9047619 ]
```

In [134…

```python
import numpy as np
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
import xgboost as xgb

# Preprocess the features
numeric_features = ['min_price', 'max_price']
categorical_features = ['commodity']

# Scale the numeric features
numeric_transformer = StandardScaler()

# One-hot encode the categorical features
categorical_transformer = OneHotEncoder(sparse=False)

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])

# Fit the preprocessor on the entire data
preprocessor.fit(X)

# Transform the testing data (including the first row)
X_test_preprocessed = preprocessor.transform(X_test)

# Extract the first row from the preprocessed testing data
selected_row = X_test_preprocessed[0]

# Reshape the row to match the model's input shape
selected_row_reshaped = np.reshape(selected_row, (1, -1))

# Define the XGBoost regressor
model = xgb.XGBRegressor()

# Train the XGBoost regressor on the entire data
model.fit(preprocessor.transform(X), Y)

# Make the prediction using the first row
prediction = model.predict(selected_row_reshaped)

# Display the prediction
print("Predicted modal price:", prediction)
```

```
Predicted modal price: [25.977482]
```

In [135…

```python
X_test
```

Out[135]:

	commodity	min_price	max_price
1760	1	0.275397	0.506581
3326	0	1.619623	1.536581
1770	1	1.283566	1.349308
3176	0	-0.665561	-0.835540
2099	0	2.291736	2.285672
...	...	...	...
2510	0	-0.262293	-0.710692
2752	0	-0.060659	-0.523419
1869	1	-0.665561	-0.991601
423	1	-0.665561	-0.835540
2990	0	-0.732772	-0.523419

800 rows × 3 columns

In [ ]: